Interoperability of Dissimilar Visual Systems: An Enhanced DIS Scene Manager

Final Technical Report

October 23, 1996 REV A (CHANGES NOTED WITH CHANGE BARS)

Contract No. N61339-95-C-0070
CDRL Sequence No. A003

Prepared for
Naval Air Warfare Center Training Systems Division
12350 Research Parkway
Code 27331/Contracts Department
Orlando, Florida 32826-3234

Prepared by
LOCKHEED MARTIN CORPORATION
Information Systems
12506 Lake Underhill Drive
Orlando, Florida 32825-5002

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Distributed Interactive Simulation (DIS) technology is maturing rapidly. Large scale distributed simulations, once too difficult to accomplish, are now becoming a reality. There are still many issues which must be resolved before DIS can function in accordance to its vision. One such issue is that of being able to interoperate within dissimilar simulator environments. DIS allows exercise environments to be constructed with very dissimilar simulators and visual image generator systems. Visual interoperability is the achievement of highly correlated rendered images on heterogeneous imaging systems. The problem of visual interoperability is specifically how to achieve it. STRICOM has sponsored investigations into mechanisms and strategies which can assist in achieving visual interoperability within DIS exercises. One mechanism for interoperability which STRICOM has sponsored is that of a Collective Scene Manager (CSM). The progress of the design and results of an investigation into the feasibility of a CSM within a DIS exercise are presented here.

2. BACKGROUND

Collective Scene Management is the management of visual information within a DIS exercise for the purpose of achieving an adequate level of correlation between dissimilar or heterogeneous visual systems. One area within image generator systems that is particularly detrimental to interoperability is that of load management. Image generator systems run the risk of crossing situations where there is too much information to process within a frame time. In many cases, the solution to this load problem is to reduce the information which results in omitted objects within the scene. Each image generator within the exercise, potentially, handles overload situations in a unique manner. Since each machine is excluding information under local constraints, correlation within the distributed exercise can be compromised. This correlation problem might be lessened or solved if a standalone server within the exercise were to monitor exercise information and predict loads within each image generator on the network. These predicted loads could then be utilized by this server to forecast overload situations within the exercise and broadcast PDUs to the participants which recommend overload avoidance measures which all participants could employ. This is the area of concentration for the Collective Scene Manager.

The Collective Scene Manager is an algorithmic suite which began in 1994 as a study of interoperability of dissimilar visual simulators in a DIS environment. This study was sponsored by STRICOM. The approach started therein was to create a DIS node within the network whose sole purpose was to gather DIS information and predict image generator polygon loads. When provided with simulator capabilities, the CSM could predict the possibility of a simulator overload situation within the exercise and, in a controlled nature, inform the simulators on how to prevent the overload. Since, under normal circumstances, each image generator system would attempt to reduce its processing load based on its own criteria, a controlled reduction across the entire exercise by a CSM would provide some degree of scene correlation for DIS connected systems.

In the phase I study, Lockheed Martin developed a prototype CSM. The prototype CSM was embedded and tested within Lockheed Martin's DIS Testbed which includes subsystems as shown in Figure 1 below. Each component of this figure is described briefly in the following sections.

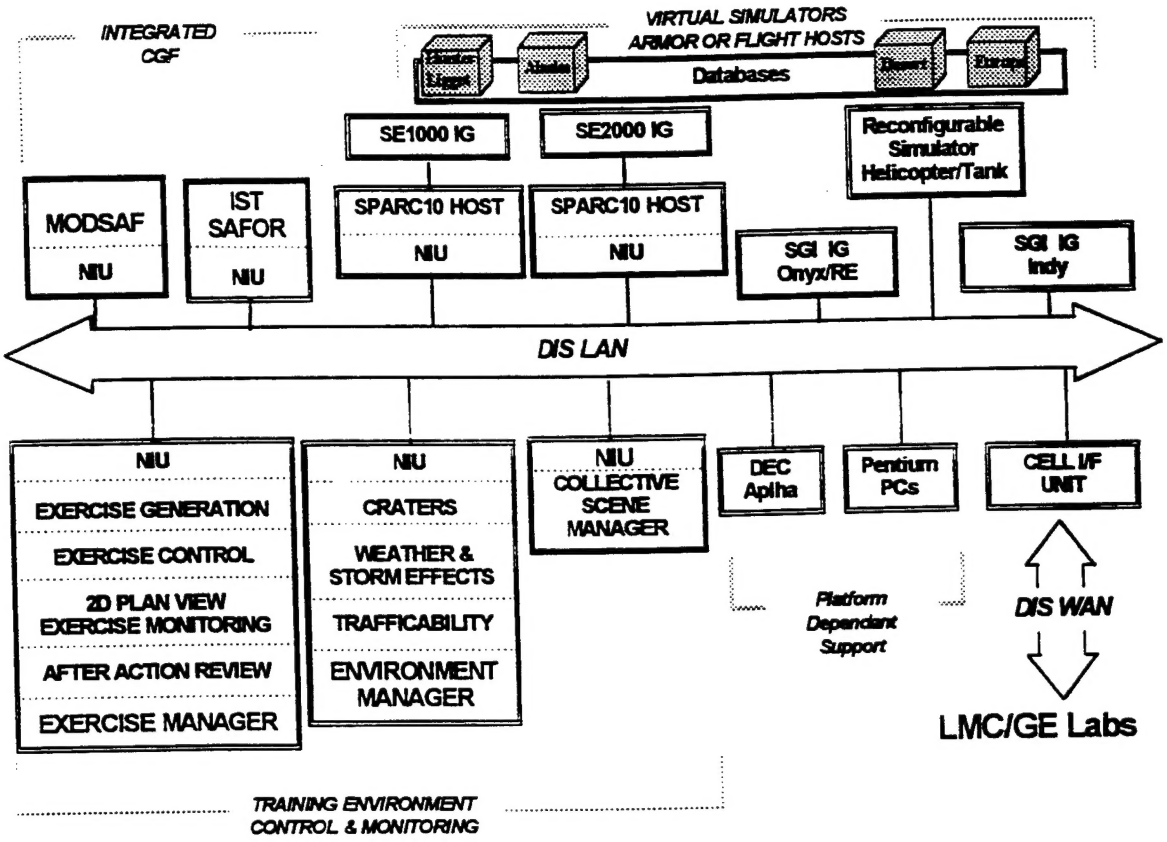


Figure 1 Lockheed Martin DIS Testbed

2.1 DIS Testbed Subsystems

The Compu-Scene SE1000 and SE2000 are real-time, z-buffered, fixed cycle, image generators. They have an extensive array of real-time, load management controls that can be accessed by a DIS based host.

The SGI Reality Engine and Indy platforms serve as real-time, z-buffered image generators. They serve as a second architecture for algorithm evaluation and are fully programmable and, therefore, amenable to real-time load management controls. Mak Technologies' VR-Link software can be utilized to link the SGI to the DIS LAN, and to provide a stealth front end to SGI's Performer software.

Several databases support the image generators within the testbed. These databases vary in both geographic location and type. The Hunter Liggett database is a discrete terrain database that is used for most DIS tests which contain diverse company participation. This was the database which was distributed in SIF format for early DIS trade shows. The Alaska database is a portion of the Interoperable Visuals/Sensors for Air Combat Command (IVACC) database and is developed primarily for flight scenarios. The desert and Europe databases are tank driver and gunner databases and contain discrete terrain as well.

This phase of CSM effort employs the database which best illustrates the interoperability issue under study. The Alaska database, provides the most appealing characteristics of all the databases within the DIS testbed. It exists in both continuous terrain format, for flight scenarios, and discrete terrain format for ground scenarios. Both forms of terrain run on the Lockheed Martin SE series of image generators. This combination of different database formats operating on the same hardware configurations creates an environment which should yield many interoperability problems.

The Alaska database is approximately a 30 square nautical mile database located in the southern portion of the Alaskan mainland. It is situated near Anchorage and includes both shoreline terrain as well as mountain ranges with elevations ranging from sea level, or zero feet, to four thousand feet. This extremely rough terrain also contributes to potential interoperability issues to study. Scenarios have been defined using this database which include ground and air vehicles interacting in combat situations, all of which can be used as needed to study interoperability.

There are two semi-automated forces program packages within the testbed which can be employed to increase the number of participating entities. IST's SAFOR can be loaded onto either of the two Pentium PCs within the DIS network. ModSAF, a more diverse semi-automated forces package which employs artificial intelligence technology to control the forces behavior, can be run on a SUN workstation or an SGI platform on the net.

Integrated into the testbed are several platforms which can be used to incorporate outside vendor software, such as ModSAF. These platforms include Pentium PCs, SGI platforms, Sun Workstations and DEC Alpha workstations. With this integrated network, software which needs to be tested or used in studies can be installed on the specific target machine it was designed for and run with very little integration effort.

There exists within the testbed, two algorithm suites, developed under Lockheed Martin IR&D efforts, which can be employed in the DIS exercises to enhance the scenarios. These algorithms are the Exercise Manager and the Environment Manager.

The Exercise Manager monitors and controls the training exercise from an observer's point of view. It contains a plan view display (PVD) user interface and performs three major functions. These functions include a scenario generation module, an after-action review, and an exercise control function. The scenario generator is used for planning exercises. Vehicle types and paths can be input through the UI. This input governs their behavior throughout the exercise. The after-action review function

gathers data during an exercise and, at the discretion of the operator, can present any or all of this data in the form of an after action review (AAR) package. This data includes such items as PVD snapshots, firing and disabling statistics, doctrinal quotes governing participant behavior, and many other items. The exercise controller provides simple exercise control mechanisms, such as starting and stopping entities during the simulation.

The Environment Manager provides the means for introducing changes in the environment into the simulation. Because of the vast nature of managing the "environment", its scope of control is limited to 3 areas of interest. It has the ability to control terrain changes by introducing artillery craters or bomb craters into the simulation. It can control weather by introducing storms or visibility changes into the simulation. It also has the ability to control mobility, and therefore vehicle dynamics, by monitoring and changing trafficability elements such as soil moisture content.

The Cell Interface Unit or CIU connects the Lockheed Martin DIS testbed to outside testbeds such as other Lockheed Martin facilities and GE Labs. DIS PDUs can be sent, back and forth, between two networks through the CIU. A Breeze 1000 Bridge/Router/Modem and a Sun Workstation serve as the CIU for this system.

As shown in Figure 1, the phase I configuration designates that the CSM run on its own Sun SPARCstation 10 connected to the DIS Local Area Network (LAN) via Ethernet. Under phase II, this function has been integrated into the Exercise manager to take advantage of the PVD.

2.2 Phase I Prototype Collective Scene Manager

The prototype CSM, developed in 1994, focused on load managing moving models by providing centralized control over multiple factors which simultaneously influenced the scene content of the connected systems. The prototype CSM successfully demonstrated the capability to control scene loading based on a combination of user controlled parameters such as moving model priority, area-of-interest designation, and field-of-view variation.

The primary purpose of collective scene management is to preserve important scene features when a simulator's visual system (IG) is faced with a database overload condition. High priority moving model entities are designated as the important scene features. CSM uses look-ahead prediction algorithms to react to potential overload conditions before they occur. All participating simulators should rank mission entities using similar values of priority, or importance to the mission. These most important models will remain in the scene in their most detailed graphical representations for all simulators, regardless of fidelities. This is an important step toward achieving a "fair game" among simulators. The major CSM functions of the prototype are shown below with a brief discussion of each function following. For a more detailed description of the prototype CSM design, the reader is referred to the "Interoperability Of Dissimilar Visual System, A Prototype DIS Scene Manager Final Results Report"(11).

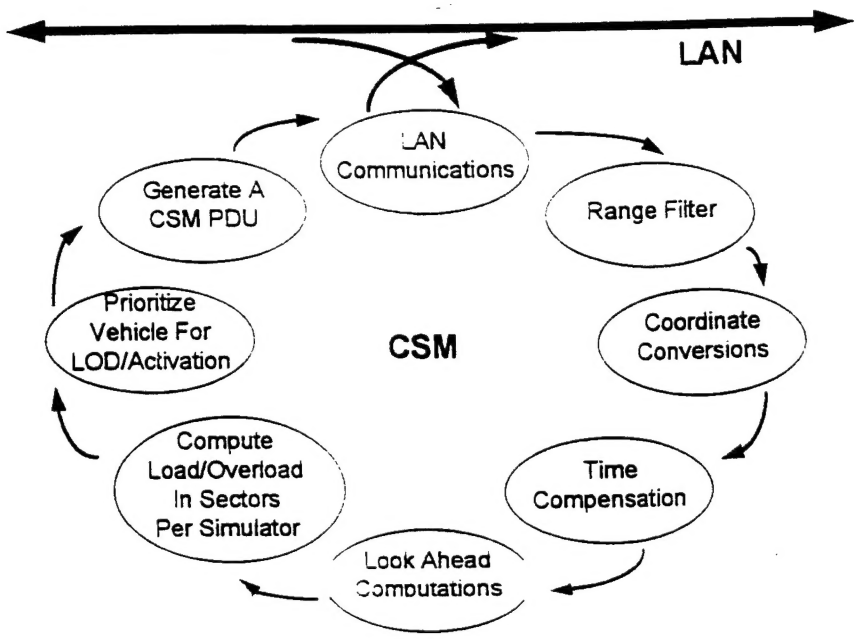


Figure 2 CSM Functional Block Diagram

The functional block diagram presented in Figure 2 illustrates the general steps which the phase I CSM took to reduce load on all simulators. The CSM begins its process by receiving and processing Entity State PDUs from the UDP/IP DIS network. A range filter is used in the next step to eliminate processing of all entities sufficiently far from the area being monitored by the CSM. Coordinate conversions are then used to convert from the DIS geocentric world to a flat earth system and vice versa. After the coordinate conversion, a time compensation algorithm is applied to correct timing errors that result from PDUs remaining in a receive queue for an excessive time or PDUs being delayed due to network collisions when there is excessive network activity. The CSM then begins to predict potential overload conditions and recommend corrective actions by looking several seconds ahead in time. It accomplishes this by extrapolating viewpoint and moving model positions. Since the future viewpoint azimuth is unknown, the CSM must process the entire horizontal scene at this predicted future position and estimate load for each individual angular sector having, in angular size, an entire field of view. Upon completing this analysis for each angular sector, the final step in the process is to generate a CSM PDU and send it to the rest of the participants. This phase I algorithm design was demonstrated in the DIS testbed as part of the prototype development.

3. SCOPE

The overall objective of the phase II CSM project is to demonstrate the application of load management techniques as an effective means of visual scene correlation for DIS connected systems. Load management techniques employed in this study are comprised of terrain loading techniques combined with load management of moving models. Since the prototype CSM investigated load management of moving models, the focus of this effort is to expand the capabilities of the prototype CSM to include dynamic allocation of polygons between terrain and model scene elements.

Several tasks must be accomplished to meet this objective. First, since the CSM works in a predictive mode, an off-line analysis of terrain loading is necessary to allow CSM to predict areas of potential overload attributed to terrain. Second, to ensure that interoperability issues do not arise due to inconsistent mission function computations, the design and development of a common mission function module (CMF) will be developed to be used on each simulator. The CMF removes any interoperability issues which may be caused by inconsistent calculations between hosts within the exercise and allows interoperability problems to be caused predominantly by rendering and load management mechanisms. This allows the CSM study to focus on its area of concentration which is rendering and load management. Furthermore, to help identify and detect load management and interoperability issues, the prototype CSM will no longer be a stand alone server, but will be integrated into a plan view display (PVD). This integration will provide the CSM with access to the visual database to assist in determining terrain loads. The PVD will also provide the mechanisms for assessing effectiveness on interoperability.

These enhancements of the CSM and algorithm development of a CMF are the preliminary and necessary tasks needed to perform the CSM studies which will assess the CSMs effectiveness on interoperability and load management. There will be three studies during this phase. The first study will demonstrate and determine the effectiveness of the CSM to load manage terrain of both discrete and continuous types. The second study will determine the effectiveness of modifying continuous terrain levels of detail on interoperability issues. This type of terrain presents the ability to achieve similar levels of detail between simulators while minimizing polygon loads. The final study will investigate the effects of using multiple distributed CSMs within the same exercise. The desired outcome of this study are insights addressing mechanisms for implementing multiple CSMs within the same exercise.

4. COLLECTIVE SCENE MANAGER (CSM)

The Collective Scene Manager, or CSM, participates in exercises on a Distributed Interactive Simulation (DIS) Local Area Network (LAN). The CSM monitors and controls one or more simulation applications. An application in this project is a stand alone simulator consisting of a Host and an Image Generator (IG). A host sends and receives Protocol Data Units (PDUs) over the DIS network. A host also reads crew station controls, performs own vehicle dynamics computations, dead reckons remote

entities, updates instrument displays, and sends commands to an image generator. The IG renders a scene showing terrain, trees, buildings, moving models, and special effects such as explosions.

The CSM sends and receives Simulation Manager PDUs, often referred to as SIMAN PDUs. The CSM uses three of the twelve SIMAN PDUs; it sends Data Query and Set Data PDUs and it receives Data PDUs. The CSM also uses data from Entity State PDUs transmitted by Simulators and Computer Generated Forces (CGF). The configuration for CSM and associated simulators is shown in Figure 3.

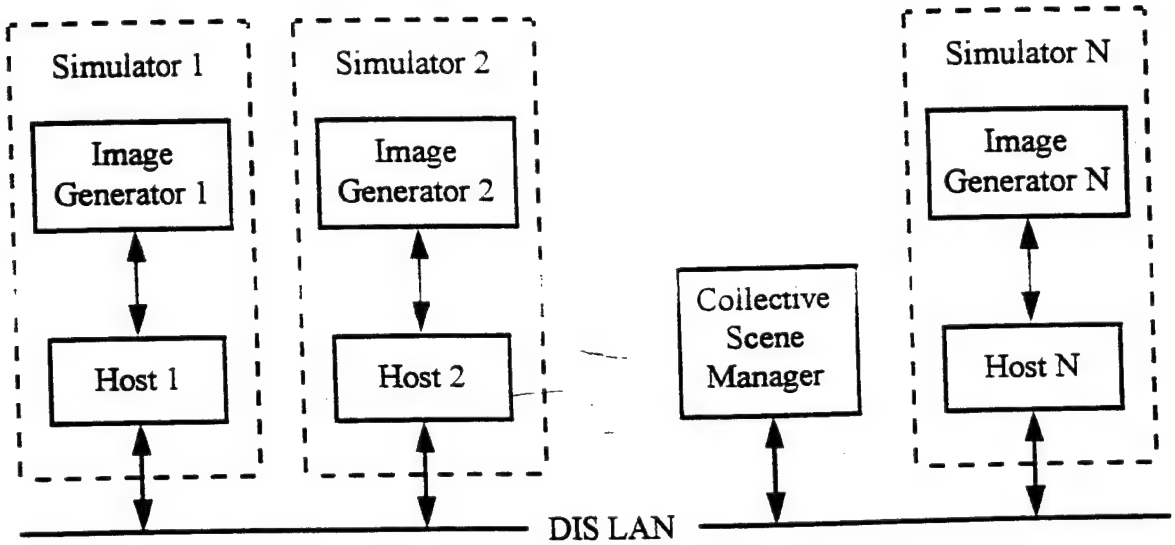


Figure 3 CSM - Simulator Configuration

4.1 Control Mechanisms

The CSM estimates the near future processing time load of the simulators that it is assigned to. If a simulator requires a processing time approaching or exceeding its frame cycle time, it is said to be operating in an overload or near overload state. The CSM attempts to avoid this condition by proposing cutbacks in the simulator's scene content. Specifically, the CSM calls for reductions in detail for selected moving models and for terrain. Reductions in level of detail (LOD) or face count allow an image generator to render a scene more quickly. The consequence of this reduction is a lower fidelity scene. The CSM will restore higher fidelity models when the threat of overload is diminished. Also, the CSM uses a priority scheme to retain higher fidelity models for entities or terrain deemed more important in the battlefield scenario.

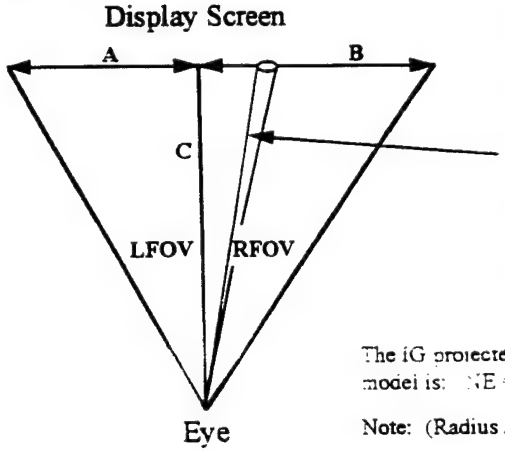
The level of detail of a model or of terrain describes its complexity. This is usually expressed as the number of faces required to render the object so that it can be identified and so that it would appear as it would in the real world. As an object is moved away from the viewpoint it gets smaller as do the faces making up the object. As faces get very small, they tend not to be distinguishable even in the real world. Therefore, they need not be displayed. This action saves IG processing clock cycles. For this reason, objects are modeled in several LODs and displayed objects are transitioned from finer LODs to coarser LODs as the objects appear farther away from

the viewer. The distance at which an object begins to change from one LOD to another is called the transition range for that LOD. Various simulators use different mechanisms for achieving this. But, as far as the CSM is concerned, all objects are LOD controlled by range ring distances. The CSM exerts further control by changing these range ring distances. It does this by requesting that each range ring distance be multiplied by a range scaling factor. The processing time change resulting from this is estimated by the CSM through use of predefined tables.

On the simulator host side, there are several control mechanisms used in scene load management. The host control algorithms for the SE 1000/2000 simulators used on this project are discussed in subsections that follow. The role of the CSM is also included.

4.1.1 Moving Model LOD Control using 3-D Blending

To understand how the SE 1000/2000 host controls moving model level of detail, an introduction to some image generator geometry must first be presented (refer to Figure 4). Consider a simulator viewer looking at a display device a distance. C from the viewer. The horizontal extent contains a number of pixels spread over a distance A + B. [Typically A = B for CRT displays.] The horizontal field of view is the sum of angles, LFOV and RFOV (left and right fields of view).



$$\tan(\text{LFOV}) = \frac{A}{C} \qquad \tan(\text{RFOV}) = \frac{B}{C}$$

$$\text{size of one pixel} = \frac{A + B}{\text{screen pixels}}$$

gamma = angle subtended by one pixel at eyepoint

$$\tan\text{ gamma} = \frac{\text{size of one pixel}}{C}$$

$$\tan\text{ gamma} = \frac{\tan(\text{LFOV}) + \tan(\text{RFOV})}{\text{screen pixels}}$$

The IG projected image size, NE (number of pixel elements), of a sphere bounded model is: NE = K (Radius / Range) ==> K = NE / (Radius / Range)

Note: (Radius / Range) is "similar" to A / C or tan gamma.

When the range is such that NE is one pixel (radius is 1 pixel), the nominal size for elimination of model from scene,

$$K = 1 / (\tan\text{ gamma}) = \frac{(\text{screen pixels})}{\tan(\text{LFOV}) + \tan(\text{RFOV})}$$

$$NE = \frac{(\text{screen pixels})\ (\text{Radius})}{(\ \tan(\text{LFOV}) + \tan(\text{RFOV})\ )\ (\text{Range})}$$

As Range decreases, NE increases causing the model to transition to finer levels of detail. If Range is multiplied by a CSM scale factor (less than one), the minimum blend size, NEmin, is increased, causing the whole LOD scale to shift. This is equivalent to pulling in "range rings".

Figure 4 3-D Model Blending

The host can control model LOD by changing the value of NEmin at which the model is deleted from the scene. This value is referred to as the *minimum size* for 3-D blending. The CSM sends a range scale factor to the host. The host divides NEmin by this factor to come up with a new minimum size that is to be sent to the IG. The host does not send this minimum size to the IG right away. It instead uses a CSM supplied

transition time over which it sends a series of minimum sizes ranging from the original size (at the time of receiving a new PDU from the CSM) to the new target minimum size. This allows the LOD transition to appear blended over time rather than abrupt.

The final issue to address, in this section, is how CSM determines a new range factor to force a change in model LOD. The CSM considers models to lie in spaces between range rings which correspond to LODs. When an overload is imminent, and a low priority model can be simplified by changing from one LOD to the next coarser LOD, the CSM calls for this transition by scaling the range rings associated with the model. The model itself is not moved. The amount of scaling is that which will place the model in the center of the space between range rings. This range scaling is illustrated in Figure 5.

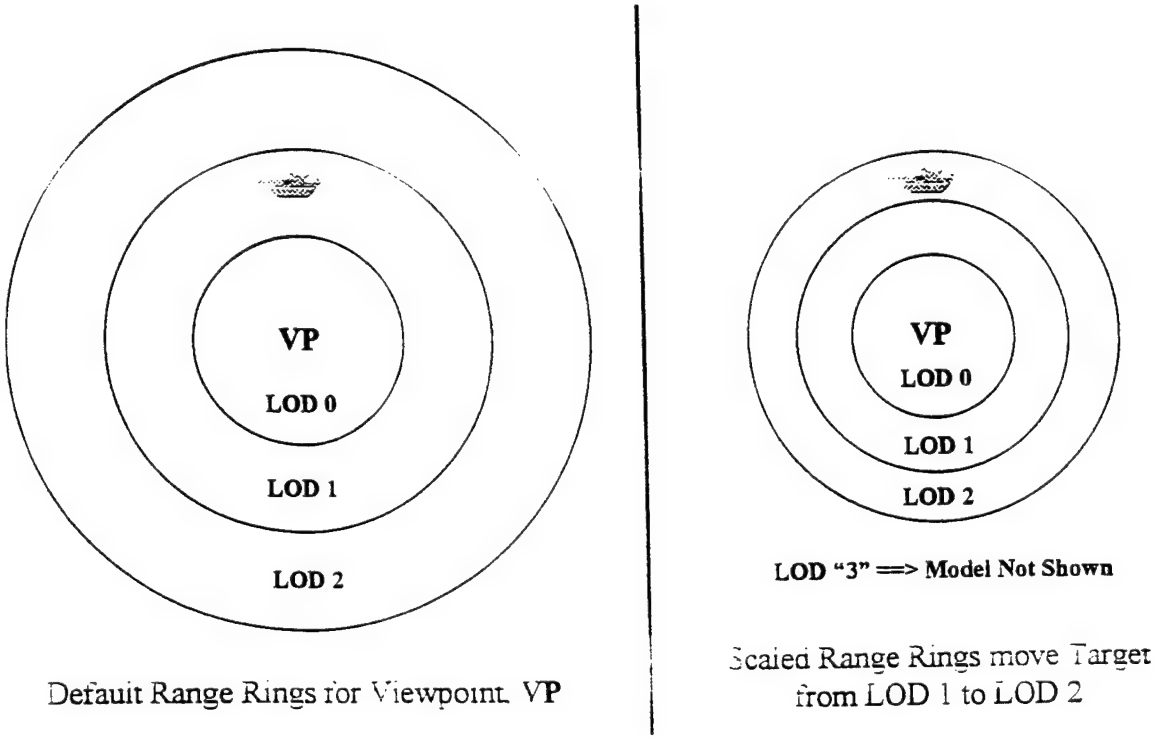
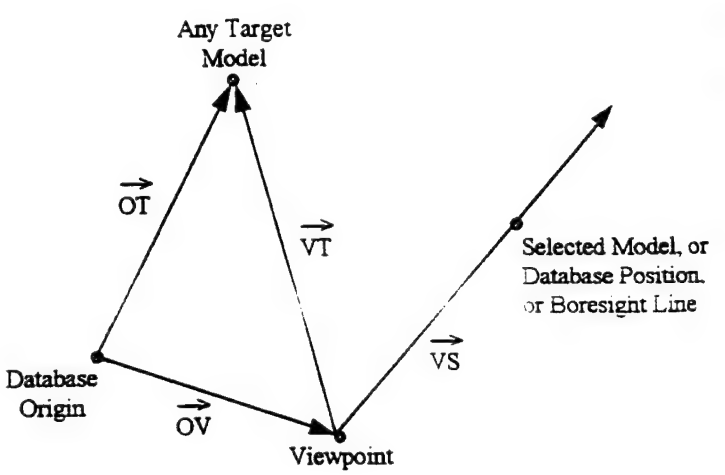


Figure 5 Range Scaling For Moving Models

The LOD transition ranges provided by the simulator during initialization should be matched to actual ranges determined by experiment.

4.1.2 Angular Control of Moving Models

The CSM can request that the Host perform continual adjustments to level of detail for all entities in the view display based on the entities' angular distances from a vector. The vector may be one of three different types: boresight, viewpoint to selected entity, or viewpoint to selected database position. The pertinent geometry is illustrated in Figure 6.



From known positions, compute vectors shown. Project them to horizontal and normalize.

Compute Dot product: $\overrightarrow{VT} \bullet \overrightarrow{VS}$

This is the cosine of the angle between the two vectors.

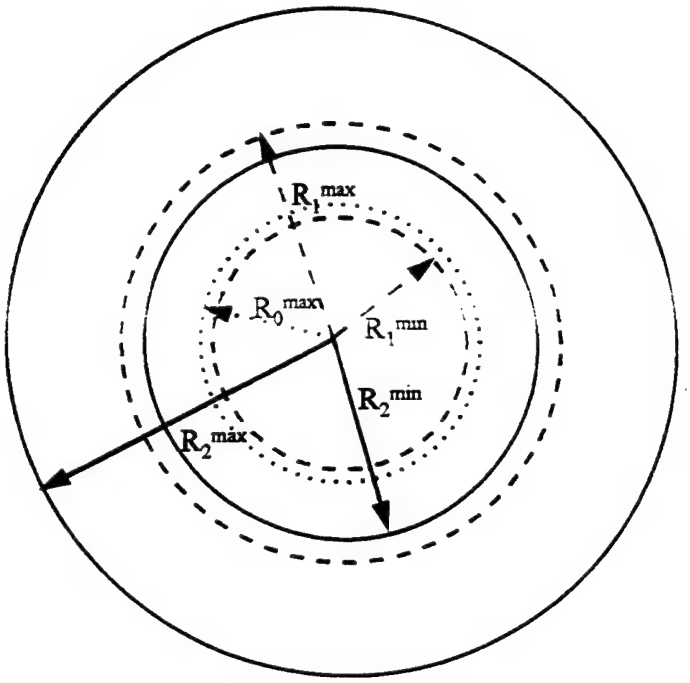
Raise this value to a user selected (or default) value. This adjustment will be divided into the minimum blend size to cause targets farther from $\overrightarrow{VS}$ to be simplified more via LOD change.

Figure 6 Vector Geometry For Angular LOD Control

When angular control is activated, entities that are far from a direction of concentration are simplified or eliminated from the scene. A practical example of this usage is an exercise where enemy targets on the other side of a river must cross a bridge in order to threaten friendly forces. The vector of concentration is from viewpoint to bridge, and thus entities near the bridge are of greatest interest.

4.1.3 Discrete Terrain Range Ring Control

Terrain databases used on this project contain multiple levels of detail. Terrain faces in the near vicinity of the viewpoint appear in greatest detail. Terrain faces that are far away are larger and offer lower fidelity. Terrain LOD regions are organized in concentric rings surrounding the viewpoint. Each LOD has a minimum and maximum range as shown in Figure 7.



$R_0^{min} = 0$ Fine Detailed Terrain appears between R_0^{min} and R_0^{max}

Medium Detailed Terrain appears between R_1^{min} and R_1^{max}

Coarse Detailed Terrain appears between R_2^{min} and R_2^{max}

In transition areas of overlap two LODs exist at once with blending applied.

Figure 7 Terrain LOD Range Rings

The minimum and maximum ranges for each LOD can be changed by the host. In particular, the host can multiply each range by a scale factor provided by the CSM. The CSM determines when the range rings should be scaled back or re-elongated based on process load conditions and priority. The gains or penalties in process load time are determined ahead of time and placed in a grid table based on viewpoint position in the database. The priorities are also decided in advance, but may be changed by the CSM operator to meet the needs of different battlefield scenarios. The terrain competes with moving models. For example, it may be decided that T72 enemy tanks are more important than terrain which is in turn more important than U.S. trucks. In an overload situation, the trucks in the scene will be first scaled back to show simpler, lower fidelity models. If these changes are not sufficient to avoid overload, the terrain range rings will be pulled in. The high priority T72s will be maintained at a high fidelity LOD as long as possible, until the scene is so complex that all entities have to be compromised to prevent an overload. This process is further complicated by the use of multiple levels of detail. For example, a truck may be simplified from LOD 0 to LOD 1 and then to LOD 2 before terrain is scaled back, but then the truck will not be simplified further (LOD 3) until after the terrain has been scaled back.

4.1.4 Continuous Terrain Error Function Control

An IG using a continuous terrain database is handled by the CSM in essentially the same way an IG using a discrete terrain database is handled. However, the host to IG mechanism deals with error Vs range functions instead of model sizes or range rings. This host control function was straight forward to implement. CSM sends a Set Data PDU containing a recommended error / range value to the host. The host then sends this same value in a small record in the interface buffer to the IG.

4.2 Communications

The CSM and simulator hosts communicate with one another by sending DIS PDUs over an Ethernet-based Local Area Network using UDP/IP (User Datagram Protocol / Internet Protocol). The CSM, which, for this project, exists as part of the Exercise Manager, uses the Network Interface Unit (NIU) software library developed at Lockheed Martin Information Systems Company (LMISC) to send and receive PDUs. LMISC's Reconfigurable Host contains a segment called DIS IO which is used for DIS communications. The communications are illustrated in Figure 8.

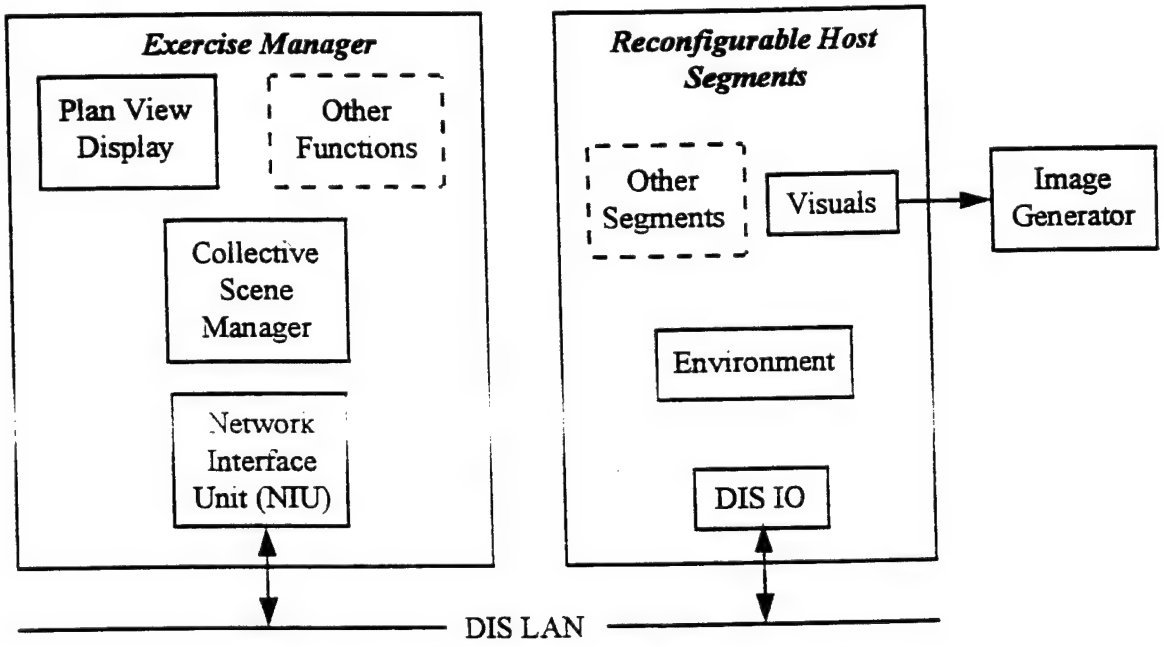


Figure 8 CSM Simulator Communications

The CSM uses DIS PDUs to request information from simulators and to control process loading on simulators' image generators. The data communicated in these PDUs are described in the subsections that follow.

4.2.1 Data Query for Simulator Initialization Data

The CSM makes use of a callback mechanism in the Exercise Manager's NIU to be notified whenever an Entity State PDU is first received from an entity not previously accounted for. If this entity is also the first received from an application, the CSM will send a Data Query PDU to that application, requesting information that the CSM will need to monitor and control that application. The Data Query PDU, with sample values in the rightmost column, appears as shown in Figure 9.

PDU Header	Protocol Version	8 bits	4
	Exercise ID	8 bits	1
	PDU Type	8 bits	18
	Protocol Family	8 bits	5
	Time Stamp	32 bits	
	PDU Length	16 bits	52
	Padding	16 bits	0
CSM Entity ID	Site	16 bits	78
	Application	16 bits	17
	Entity	16 bits	0
Host Entity ID	Site	16 bits	78
	Application	16 bits	28
	Entity	16 bits	0
Request ID (unique for this PDU)		32 bits	123
Time Interval (0 ==> answer once)		32 bits	0
Number of Fixed Datum Records		32 bits	0
Number of Variable Datum Records		32 bits	3
Variable Datum ID (IG Properties)		32 bits	214790110
Variable Datum ID (Terrain Properties)		32 bits	214790115
Variable Datum ID (Moving Model Properties)		32 bits	214790120

Figure 9 Data Query for Simulator Information

The PDU header format is standard for all DIS PDUs. It contains six items plus padding. Protocol version 4 corresponds to standard 2.0.4 or more formally IEEE 1278.1.94 which was used by all participants of I/ITSEC 95. The exercise ID, set to 1 in this example, should be common for all applications for a joint mission. The PDU type number for a Data Query PDU is 18. The SIMAN PDU family has been designated to be 5. The time stamp would contain the time past the hour that the PDU is transmitted. The sample PDU has a length of 52 bytes.

The Exercise Manager with the CSM as an integral part, and a host simulator are run on a workstation that are given DIS addresses. All of the participants in the LMISC DIS lab use site 78. Application addresses such as 17 and 28 above are assigned uniquely to resident PCs and workstations. The applications themselves are not entities, and hence use entity ID = 0.

The request ID value, 123 comes from a static counter in the Exercise Manager. The responding Data PDU (see section 4.2.2) should use this same value. A time interval value of 0 tells the host to return requested information one time only.

The data being requested will contain no fixed datum records and 3 variable datum records. The records will contain IG properties, terrain properties and moving model properties. These will be discussed in section 4.2.2.

4.2.2 Simulator Data Response to (Initialization) Data Query

A simulator receiving the Data Query PDU discussed above will respond by sending the CSM a Data PDU containing data for each Datum ID mentioned in the Data Query PDU. Actually, data for many moving model types may be included even though only one Datum ID is in the Data Query PDU. The CSM has no way of knowing how many model types are supported by any particular simulator. Figures 10-13 show a typical Data PDU sent from Host to CSM.

PDU Header	12 bytes	
CSM Entity ID	6 bytes	
Host Entity ID	6 bytes	
Request ID (equal that of Data Query PDU)	32 bits	123
Padding	32 bits	0
Number of Fixed Datum Records	32 bits	0
Number of Variable Datum Records	32 bits	3
Variable Datum ID (IG Properties)	32 bits	214790110
Variable Datum Length (bits)	32 bits	128
Field of View of Channel (degrees)	32 bits float	20.0
Frame Cycle Time (seconds)	32 bits float	0.033
Horizontal Screen Size (pixels)	32 bits	1024
Channel Number	16 bits	0
Number of Entity Types Supported	16 bits	1
Variable Datum ID (Terrain Properties)	32 bits	214790115
[Examples Expanded in Next 2 Figures]		
Variable Datum ID (Moving Model Properties)	32 bits	214790120
[Expanded in Third Figure Below]		

Figure 10 Simulator Information Data PDU (1 of 4)

The first variable datum record contains IG properties. As in all variable datum records, the ID of the variable datum record and the size of the included data in bits (128) are first. The IG display channel uses a horizontal field of view of 20 degrees and contains 1024 pixels. The IG operates at 30 Hz and thus has a frame cycle type of

0.033 seconds (33 milliseconds). The IG channel number is designated as 0. The simulator supports one entity type. The other two datum records are shown in separate figures.

Variable Datum ID (Terrain Properties)	32 bits	214790115
Variable Datum Length (bits)	32 bits	512
Terrain Type (0 =>Discrete, 1=> Continuous)	32 bits	0
Number of Scale Levels	32 bits	3
Priority of Terrain	32 bits	40
Number of Range Rings (for Discrete Terrain)	32 bits	3
1st Scale Factor Value	32 bits	1.00
1st Scale Priority	32 bits	1.00
2nd Scale Factor Value	32 bits	0.85
2nd Scale Priority	32 bits	2.00
3rd Scale Factor Value	32 bits	0.70
3rd Scale Priority	32 bits	3.00
1st Range Ring Minimum (feet)	32 bits float	0
1st Range Ring Maximum (feet)	32 bits float	30000
2nd Range Ring Minimum (feet)	32 bits float	25000
2nd Range Ring Maximum (feet)	32 bits float	100000

Figure 11 Simulator Information Data PDU (2 of 4)

Discrete terrain properties (Figure 11) include a priority value to allow terrain to compete with vehicles for display fidelity. In this example, the terrain has 2 range rings with LOD transitioning occurring between 25000 and 30000 feet. The CSM will have the opportunity of scaling back these ranges to 85% or 70% of these distances. The priorities will rise from 40 to 80 or 120 for these cutbacks in scene level of detail using the multiplicative scale priority factors (1.0, 2.0, and 3.0).

Alternatively, continuous terrain properties (Figure 12) may appear in the Data PDU from the Host. Terrain priority is handled in the same way as for discrete terrain. For continuous terrain scale factors are equivalent to Error/Range values that will be sent, as is, to the Image Generator to control the level of detail in the scene.

Variable Datum ID (Terrain Properties)	32 bits	214790115
Variable Datum Length (bits)	32 bits	512
Terrain Type (0 =>Discrete, 1=> Continuous)	32 bits	1
Number of Scale Levels	32 bits	3
Priority of Terrain	32 bits	40
Number of Range Rings (for Discrete Terrain)	32 bits	0
1st Scale Factor Value (Nominal Error/Range)	32 bits	19.0
1st Scale Priority	32 bits	1.0
2nd Scale Factor Value	32 bits	24.5
2nd Scale Priority	32 bits	2.0
3rd Scale Factor Value	32 bits	30.0
3rd Scale Priority	32 bits	3.0

Figure 12 Simulator Information Data PDU (3 of 4)

Variable Datum ID (Moving Model Properties)		32 bits	214790120
Variable Datum Length (bits)		32 bits	384
Entity Type: (T72)	Kind	8 bits	1
	Domain	8 bits	1
	Country	16 bits	222
	Category	8 bits	1
	Subcategory	8 bits	2
	Specific	8 bits	1
	Extra	8 bits	0
Model Priority		32 bits float	100.0
Number of Levels of Detail		32 bits	2
First LOD:	Processing Time	32 bits float	0.0015
	Transition Range	32 bits float	4200.0
	LOD Priority	32 bits float	1.0
	Padding	32 bits	0
Second LOD:	Processing Time	32 bits float	0.0003
	Transition Range	32 bits float	33600.0
	LOD Priority	32 bits float	10.0
	Padding	32 bits	0

Figure 13 Simulator Information Data PDU (4 of 4)

Moving model properties are similar to terrain properties. The T72 tank identified by entity type, 1:1:222:1:2::0 as defined in the DIS Enumerations Document [14], is given a priority of 100. This means a T72 will not be simplified in the IG until after terrain is first simplified. Had a friendly M1 tank been used instead. the model priority could have been chosen to be less than 40, causing the M1 to be simplified at least once before reducing the fidelity of the terrain during a time of IG overload conditions.

The model has two levels of detail. For each LOD three parameters are provided. The processing time is the time that the IG needs to render the model at the particular LOD. The transition range is the range (in feet) that the model would naturally change from the given LOD to the next coarser LOD. LOD priority is a multiplicative factor to apply to the model priority to get the total priority of the model at any particular LOD.

4.2.3 Set Data PDU for LOD Range Control

The CSM controls a simulator's processing load by sending a PDU with a request to change the range ring scaling for one or more moving model entities and/or for terrain faces. The Set Data PDU contents for changing the range scaling of one moving model is shown in Figure 14.

PDU Header		12 bytes	
CSM Entity ID		6 bytes	
Host Entity ID		6 bytes	
Request ID		32 bits	125
Padding		32 bits	0
Number of Fixed Datum Records		32 bits	0
Number of Variable Datum Records		32 bits	1
Variable Datum ID (LOD Range Control)		32 bits	214790010
Variable Datum Length (bits)		32 bits	160
LOD Range Control Type (0 ==> an entity) (1 ==> terrain)		32 bits	0
Entity ID	Site	16 bits	78
(0's if terrain)	Application	16 bits	10
	Entity	16 bits	5
Padding		16 bits	0
Range Scale Factor		32 bits float	0.8
Transition Time (seconds)		32 bits float	1.5

Figure 14 LOD Range Control Set Data PDU

The header structure is similar to those discussed previously as are the variable datum ID and length fields. The LOD range control type set to 0 indicates a moving model entity is to be affected. Thus, an entity ID is required to identify which specific model [(78:10:5), read as (site:application:entity) as given in Figure 14] is to undergo an LOD change. Had terrain been controlled instead (LOD range control type = 1), the entity ID would be set to all zeroes. The range scale factor, 0.8 indicates that all LOD range rings, for the entity, are to be reduced by multiplying transition ranges by that

factor. The host simulator is to perform this reduction in linear steps over the specified transition period (1.5 seconds in the example).

4.2.4 Set Data PDU for LOD Angle Control

The CSM's angular control of moving models is detailed in section 4.1.2. The CSM uses a Set Data PDU to convey its intent to the host. It is up to the host to continually check and update moving model LODs based on their angular distance from the sight vector specified by the CSM. A sample PDU for a viewpoint to entity vector is shown in Figure 15.

PDU Header		12 bytes	
CSM Entity ID		6 bytes	
Host Entity ID		6 bytes	
Request ID		32 bits	126
Padding		32 bits	0
Number of Fixed Datum Records		32 bits	0
Number of Variable Datum Records		32 bits	1
Variable Datum ID (LOD Angle Control)		32 bits	214790020
Variable Datum Length (bits)		32 bits	320
LOD Angle Control Type	(0 ==> none) (1 ==> boresight) (2 ==> entity) (3 ==> position)	32 bits	2
Cosine Exponent Power Factor		32 bits	0.6
Entity ID	Site	16 bits	78
	Application	16 bits	11
	Entity	16 bits	5
Padding		16 bits	0
Database Position: X		64 bits float	0.0
(geocentric) Y		64 bits float	0.0
Z		64 bits float	0.0

Figure 15 LOD Angle Control Set Data PDU

Following the usual header data is an LOD angle control type value that states which kind of vector is to be referenced (1=boresight, 2=viewpoint to selected entity, or 3=viewpoint to specified database position). If this control type is set to zero, it is an indication for the host to stop performing LOD angular control altogether. In the example, option 2 is selected. Thus an entity ID (78:11:5) is specified and the database position is set to all zeroes. The cosine exponent power factor is required for all three types of vectors. It is used to govern how much LOD reduction is applied to entities based on their angular offset from the reference vector.

4.2.5 Data Query For IG Load Data Feedback

When the CSM operator activates load management for a particular simulator, the CSM issues a Data Query PDU for that simulator, requesting that it return actual process time load data to the CSM. Furthermore, this data is to be continually reported to the CSM at a rate specified in the Data Query PDU. This PDU is constructed as shown in Figure 16.

PDU Header	Protocol Version	8 bits	4
	Exercise ID	8 bits	1
	PDU Type	8 bits	18
	Protocol Family	8 bits	5
	Time Stamp	32 bits	
	PDU Length	16 bits	44
	Padding	16 bits	0
CSM Entity ID	Site	16 bits	78
	Application	16 bits	17
	Entity	16 bits	0
Host Entity ID	Site	16 bits	78
	Application	16 bits	28
	Entity	16 bits	0
Request ID (unique for this PDU)		32 bits	130
Time Interval (1 second as DIS Time Stamp)		32 bits	0x123456
Number of Fixed Datum Records		32 bits	0
Number of Variable Datum Records		32 bits	1
Variable Datum ID (IG Load)		32 bits	214790130

Figure 16 Data Query for IG Load

The PDU header is similar to those shown previously. The Data Query PDU type is 18. The source, (78:17:0) identifies the computer running the Exercise Manager which contains the CSM. The destination host being queried is (78:28:0). The request identifier, 130 will allow for correlation with the Data PDU that the Host will return. The CSM specifies a time interval of one second, meaning that the host is to return the data requested once every second. The datum record requested is variable and is identified by ID, 214790130.

4.2.6 Host to CSM Data PDU Containing Process Load

The Data PDU sent, at intervals, to the CSM from a simulator host is described in Figure 17. The significant data word is the Frame 2 processing time for an SE1000 or SE2000 Image Generator. Frame 2 is the Image Generator hardware that deals with geometry processing. [Frame 1 is a general purpose computer front end to the IG, and Frame 3 is the IG pixel processor.] Other data, such as Frame 2 face count and Frame 3 processing time is also available, but the Frame 2 time is thought to be the overload bottleneck, and Frame 2 processing is the most direct benefactor of the controls available for the CSM to modify.

PDU Header	12 bytes	
CSM Entity ID	6 bytes	
Host Entity ID	6 bytes	
Request ID (equal that of Data Query PDU)	32 bits	130
Padding	32 bits	0
Number of Fixed Datum Records	32 bits	0
Number of Variable Datum Records	32 bits	1
Variable Datum ID (IG Load)	32 bits	214790130
Variable Datum Length (bits)	32 bits	64
(Frame 2) Processing Time (seconds)	64 bits float	0.024500

Figure 17 IG Process Load Data PDU

Following the usual PDU header is the variable datum record which contains just one double precision number. In the example the Frame 2 processing time is 0.0245 seconds, or 24.5 milliseconds. If the IG runs at 30 Hz, there is currently no danger of overloading the 33.3 millisecond frame rate.

4.2.7 CSM To CSM Data PDU For Multiple CSMs

In a multiple CSM exercise, one CSM called CSM A might hand off its control of a simulator to another CSM called CSM B by sending CSM B a Set Data PDU with a variable datum record as shown in Figure 18. The datum record contains some of the data that CSM B needs to reproduce to successfully monitor the simulator. The rest of the required data is contained in initialization data files and in a data record to be obtained from the simulator itself after sending it a Data Query PDU (see section 4.1.2.1). CSM B also returns a Data PDU to CSM A to acknowledge receiving CSM A's Set Data PDU and to acknowledge acceptance of responsibility for the simulator. This Data PDU is essentially a copy of the Set Data PDU and is thus also defined by Figure 18.

PDU Header		12 bytes	
CSM A Entity ID		6 bytes	78:17:0
CSM B Entity ID		6 bytes	78:14:0
Request ID		32 bits	145
Padding		32 bits	0
Number of Fixed Datum Records		32 bits	0
Number of Variable Datum Records		32 bits	1
Variable Datum ID (Simulator Handoff)		32 bits	214790500
Variable Datum Length (bits)		32 bits	192
Simulator Ownship Entity ID:	Site	32 bits	78
	Application	32 bits	28
	Entity	32 bits	5
IG Channel		32 bits	0
IG Processing Rate (Hz)		32 bits	30
Terrain Process Load Grid Data File Index		32 bits	1

Figure 18 CSM to CSM Simulator Hand-off PDU

4.3 Implementation

Individual software applications and algorithms are discussed in the subsections that follow.

4.3.1 Scene Load Management Algorithms

The run-time loop of the CSM is called by the Exercise Manager's run-time loop each cycle. The CSM loop uses a counter to limit its calls to the scene load management code. Typically, this application code is run once a second. At that time load management algorithms are invoked for each simulator being monitored by the CSM.

If a simulator has an ownvehicle and load management is enabled, then functions are called to process the following: compute extrapolated positions for the ownship and remote entities for a near future time (e.g. 5 seconds), determine if moving model or terrain levels of detail should be adjusted to avoid overload or recover from underload, and construct and send commands to the simulator in Set Data PDUs.

Linear extrapolations are performed using the most recent position data received from Entity State PDUs which are captured by the Exercise Manager and stored where they can be easily accessed by the CSM software. Ownvehicle attitude is also linearly extrapolated.

Overload and underload processing are identical in structure but opposite in objective. In overload processing lowest priority models or terrain are simplified to cut down on IG processing, while in underload processing highest priority models or terrain are made more complex to achieve a desired fidelity when the total processing load is

well under the overload threshold. Moving models are initially ranked in priority order. The terrain's place in this priority list is also determined. Models are grouped in sectors based on ownship heading angle. Multiple sectors are required because the ownship's future heading is not pre-determinable. Each sector covers an area based on the ownship horizontal field of view. In each sector model LODs are modified, one at a time, until the overload (underload) threat is avoided. The terrain LOD may also be modified during this process. However, the terrain LOD applies to all angular sectors.

The CSM computes the IG processing load of terrain using bilinear interpolation of grid values pre-determined off-line. Heading angle and terrain range ring scaling are taken into account. Processing loads for each moving model entity, based on current LOD, are found in the initialization tables provided by each simulator. The sum of all of these loads is the CSM predicted load and it is compared to the time limit to see if an overload condition exists.

CSM determined LOD changes are grouped in datum records and placed in a Set Data PDU which is sent to the simulator application. The Exercise Manager's NIU does the detailed work required to realize this communication.

4.3.2 Integration with Exercise Manager / Plan View Display (PVD)

For this project, the Collective Scene Manager has been made a part of LMISC's Exercise Manager. However, the connections between the CSM and Exercise Manager have been kept to a minimum and have been well defined. The goal is to maximize independence so that the CSM may be integrated with other systems in the future. However, at the same time, the power and flexibility of the Exercise Manager is fully utilized. In particular, its user interface and its connectivity to the DIS network are used by the CSM.

An independent Motif X window set was designed for the CSM using Imperial Software Technology's X Designer. The main CSM window is "connected" to the Exercise Manager user interface by a mode option on its main window and by a limited number of lines of code in the Exercise Manager source files. pvd.cc and exrmgr_stubs.cc. The code fragments can all be located by searching for the three characters, "CSM". Several statements were also inserted to reference the CSM main window and supporting pop-up windows.

Calls to CSM functions, "CsmInit" and "CsmLoop" are added to the Exercise Manager. CsmInit, which performs initialization duties, is called just once, when the Exercise Manager operator first chooses the CSM "mode". Run-time function, CsmLoop is called during each cycle of Exercise Manager processing if the CSM mode is active.

Also added to Exercise Manager code are setup statements for several CSM callback functions. "CsmEntityAddedCallback" is called whenever the Exercise Manager first detects the presence of a new entity on the network due to an Entity State PDU. "CsmEntityRemovedCallback" is called when an entity leaves the simulation by no longer sending Entity State PDUs. The CSM needs to do bookkeeping on these entities just as the Exercise Manager does. "CsmDataPDUReceivedCallback" is called whenever the Exercise Manager's NIU

receives a Data PDU because some Data PDUs are responses to CSM initiated Data Queries or Set Data PDUs. "CsmSetDataPDUReceivedCallback" is called when a CSM receives a request for a simulator hand-off from another CSM.

The CSM also makes calls to Exercise Manager functions. One, "disNiu->SendPDU", uses the network interface unit library to send PDUs out over the DIS network. Another, "disNiu->GetDisTime", fetches the current time or a future time and expresses this time in standard DIS time notation. The function, "pvdApp->GetCurrentEntity" is called to lock onto the entity selected on the Exercise Manager's Plan View Display. Several other pvdApp functions are called to deal with PVD drawing layers and PVD Tools which is discussed in the next section.

4.3.3 PVD Tools

As the CSM strives to deal more closely with interoperability issues directly, the need for a set of analysis tools becomes increasingly important. Tools which a user can employ to detect dissimilar scenes within the same exercise can assist in locating and documenting interoperability issues. During this effort three such tools were implemented. All three take advantage of the graphical interface which the PVD lends to the CSM.

The first tool is a line of site (LOS) contour which depicts a horizontal LOS contour around a selected object(s) within the exercise. This tool is useful when generally trying to determine if target engagement is possible. It does not, however, make a fair assessment of the intervisibility between two entities. For example, notwithstanding tunnels or overhangs, an object falling within this horizontal LOS contour is definitely visible to the entity, while an object lying outside this contour may or may not be visible to the entity.

A better method of assessing intervisibility would be a direct line of site tool. This is the second tool implemented under this study. This tool, when selected by the user, draws a single LOS vector from the selected entity to every other entity within the exercise. The LOS vector intersects an occluding object or an entity, whichever comes first, along its vector. This tool is extremely useful but can be computationally intense, depending on the size of the exercise. This tool can be of great use in visually illustrating interoperability problems. A display of LOS vectors in varying colors depicting each simulator's viewpoint to all remote entities in the exercise is possible with this tool if LOS information is provided by the simulators. An interoperability issue would occur when an entity did not have two complete LOS vectors between it and a remote entity. For the CSM study, this capability provides a means by which to detect interoperability problems. For a trainer, overseeing a DIS exercise, this tool provides a mechanism to note or visually see these interoperability problems. This tool also provides the means to automate the detection process and incorporate this observation within an after-action review.

The final tool implemented in this study is the Terrain Sector Load Tool. This tool draws a pie chart around the selected entity and annotates the terrain load

information within each slice. It requires the terrain load analysis information which is collected off-line as input.

The figures below illustrate the tools mentioned above as seen through the PVD incorporated with the Exercise Manager. Figure 19 shows the LOS contour tool. The black outline in the lower right image area shows one entity's horizontal LOS contour at a specific time during the exercise. Figure 20 shows the LOS vectors for the same exercise. Each black line is a vector which extends from the selected object to every remote entity within the exercise. Figure 21 shows the Terrain Sector Load tool. The pie shaped wedges define angular sectors and the value annotated within each wedge is the processing time that the IG requires to render just the terrain in that direction.

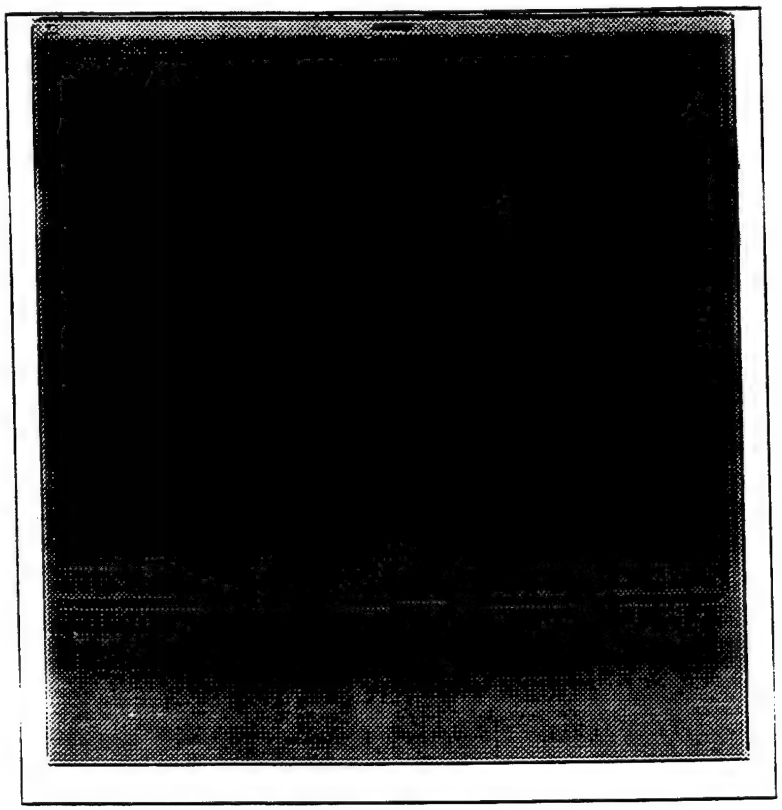


Figure 19 LOS Contour

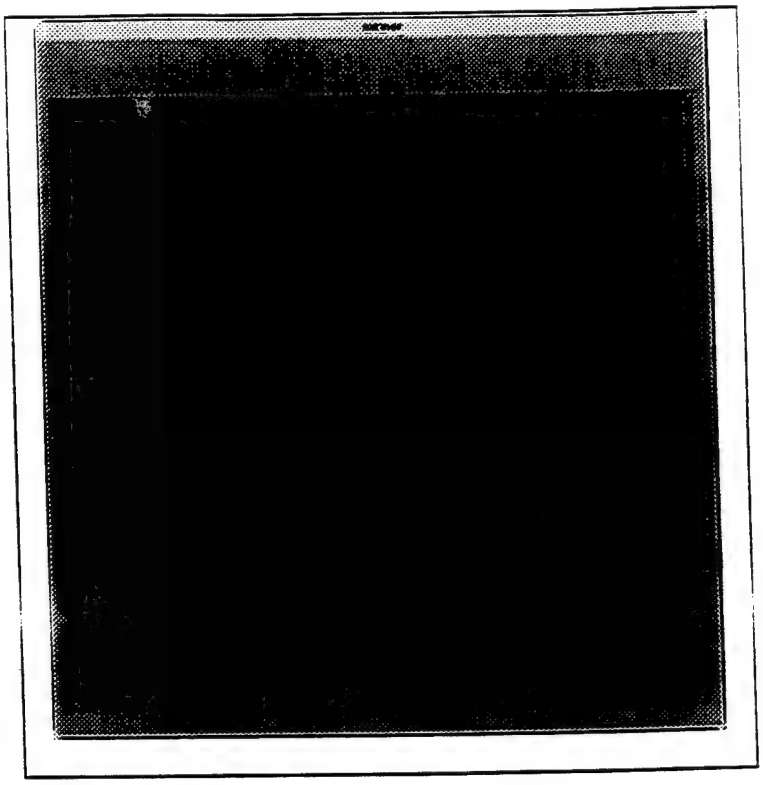


Figure 20 LOS Vectors

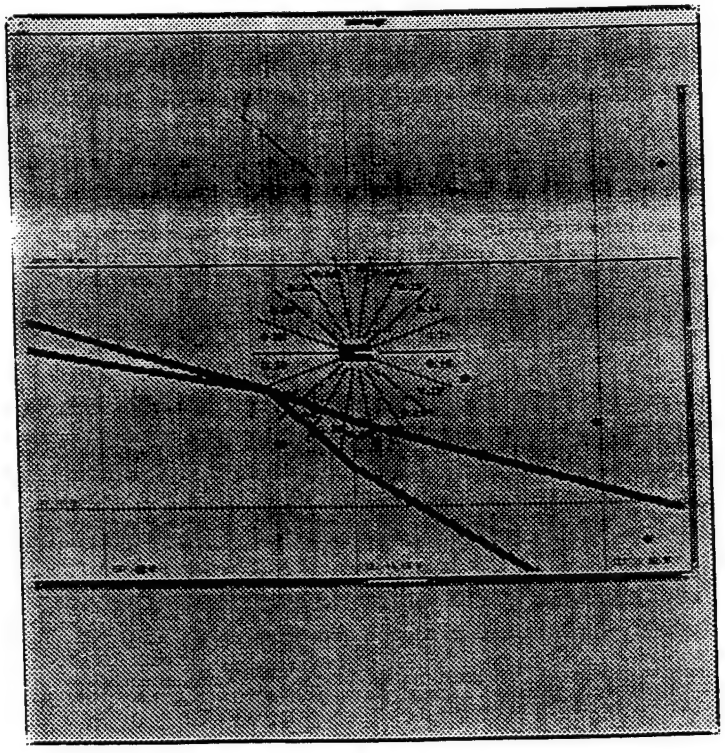


Figure 21 Terrain Sector Load

5. TERRAIN LOAD ANALYSIS

5.1 Overview

The purpose of terrain load analysis is to provide a mechanism for CSM scene load management to utilize terrain loading data in computing scene content. As part of the terrain load analysis, a grid or look-up table is developed off-line and defines the processing time of the image generator geometry processor for grid points in a desired database region. The CSM utilizes this grid to predict a potential overload situation associated with the current heading or view during run-time. If an overload condition is predicted, the CSM may elect to change the terrain range rings (discrete or blended terrain) or error function (continuous terrain) to effectively reduce the image generator processing time. This reduction in processing time, may in turn, allow for necessary high priority models to be able to be processed and included in the simulation scene without overload.

5.2 Implementation

Prior to generating the look-up table, a study was performed to determine the best measure of image generator processor time for scene load management. Geometry Processor (GP) face count and processing time were both considered, as well as pixel processing time. Since the processing load of both the GP and the pixel processor are dependent on the number of faces rendered, it would seem logical to use the number of faces as the appropriate measure of processing load. However, the face count is not always linearly proportional to the GP processing time. The reason is that the GP processor culls and throws out a different number of occluded faces each frame, thus utilizing varying amounts of processing time. Therefore, it is possible to display an identical number of polygons in two unique scenes, yet have a large variance in processing time. Since the total GP processing time consumed within a field is ultimately the most important criteria, it is a more accurate variable than face count when used in a predictive scene load management algorithm. Also, while the pixel processor time may also be a contributing factor to overload situations, the most feasible method of reducing this time is by reducing the number of polygons rendered, which is a Geometry Processor function. A reduction in polygon faces rendered causes a reduction in texture and pixel processing time. Thus, the GP processing time was used in this study in generating the terrain load look-up table.

The terrain load grid or look-up table is generated off-line by a custom designed host. This host divides the given database region into many grid points, where each data point of the grid is to be used by the CSM to predict processing time. For this study, a portion of the LMISC Alaska database was chosen. The latitudinal grid points were preliminarily chosen to be 24 arc seconds apart, while the longitudinal grid points were chosen to be 48 arc seconds apart. These grid deltas correspond to a difference of approximately 2400 feet between points. Note, the smaller the grid size, the more accurate the CSM load estimation will be.

For discrete terrain, range ring scaling factors are defined. These are used to create processing load data for all grid points where the LOD range rings are set at different ranges. The data allows CSM to change the range rings to reduce processing

load if necessary during run-time. For this study, the range scaling factors were set at full scale, 85%, and 70%.

The terrain load analysis process collects data for each grid point. The processing time of the IG geometry processor is first collected for a respective grid point with the range rings set at their default range. The data is collected for 18 viewpoint headings at the given point, each 20 degrees apart. The host then sends the image generator a command to pull in the range rings for all terrain LODs to the next defined scaling value, and the GP processing time data is once again collected at each heading. All data is stored in a lookup table format, and the process is complete when all grid points, headings, and range scales have been processed. The format of the look-up table generated in this study is found in Table 1.

Table 1 Terrain Load Table Format

LAT INDEX	LON INDEX	RANGE SCALE	0° Load (ms)	20° Load (ms)	40° Load (ms)	60° Load (ms)	80° Load (ms)	--	320° Load (ms)	340° Load (ms)
0	0	0	4.46	4.97	6.40	9.22	10.35	--	4.97	4.73
0	0	1	4.35	4.36	5.79	7.83	8.74	--	4.83	4.46
0	0	2	4.08	3.87	4.97	6.62	7.16	--	4.46	4.38
0	1	0	4.20	5.19	7.20	9.45	10.24	--	4.67	4.14
0	1	1	4.23	4.69	6.48	7.87	8.59	--	4.42	3.89
0	1	2	4.05	4.13	5.56	6.66	6.96	--	4.21	3.77
0	2	0	5.18	5.82	7.41	9.72	9.91	--	5.24	4.46
:	:	:	:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:	:	:	:
22	24	2	5.67	5.17	5.00	5.17	5.47	--	5.22	5.46
22	25	0	7.14	6.99	7.12	7.79	7.64	--	6.42	7.31
22	25	1	6.03	5.75	6.13	6.70	6.45	--	5.95	6.84
22	25	2	5.42	4.78	5.25	5.62	5.45	--	5.14	5.43

5.3 Discussion

Several issues concerning the terrain load analysis approach were discovered throughout the course of this study, some of which may be considered in future developments. One of these issues concerns CSM's use of bilinear interpolation to estimate terrain load between grid points along the same heading as the viewpoint. This is an accurate approach for most applications, but the accuracy becomes lessened when one considers a situation where a database object with several faces (such as a mountain) exists between the moving model viewpoint and a grid point, as illustrated in Figure 22. If the heading of the viewpoint is such that the large database object is outside the field of view, the bilinear interpolation will yield an estimation much higher than it should. As shown in this example, the inaccuracy results since the bilinear interpolation algorithm utilizes point B which has a very high load value. In reality, the load for this moving model will be much closer in magnitude to the load of grid point A. The inaccuracy of this approach is minimized as the grid point deltas are reduced and more grid points are used; however, this is at the expense of adding more data to an already large look-up table.

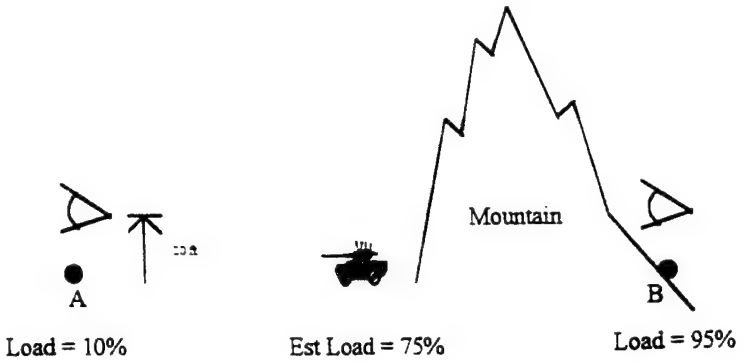


Figure 22 Bilinear Interpolation Limitation

Another issue discovered concerning terrain load analysis and the look-up table is that currently the processing load is taken at each x, y grid point where the viewpoint is placed 20 feet above the terrain at zero pitch, as shown in Figure 23. This is accurate for most flat or low slope ground applications, but becomes a somewhat poor approximation when a ground vehicle is on a steep slope. In such a case, the current terrain load analysis approach gives an estimate of the processing load 20 feet above the slope, but instead of looking in the direction of the viewpoint pitch (or terrain pitch), the load is measured as if the viewpoint is looking straight ahead. This approach provides a worst case processing load, since a viewpoint at zero pitch is more likely to see more polygon faces than a viewpoint looking into the sky or ground. It would be possible to measure grid data utilizing the pitch of the terrain, but this would not be as accurate in some cases. For example, consider the case where the processing load is measured for a grid point located on a steep hill, as shown in Figure 24. The measured processing load for grid point A is low, since the viewpoint is looking into the sky. However, when the processing load for this grid point is used in bilinear interpolation during run-time to compute the future processing load for the moving model, a low processing load value is estimated. In this example, the moving model's future field of view is in reality looking at the mountain, where a somewhat large face count should be observed. The low processing load estimation caused by using grid point A may cause an overload situation to occur. As shown in Figure 23, using a height above terrain of 20 feet in this same situation provides a more accurate terrain load estimation for the moving model.

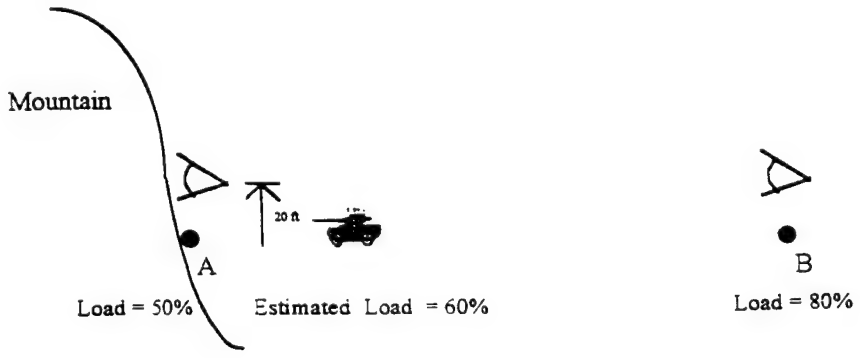


Figure 23 Terrain Load Analysis

Figure 24 Terrain Load Analysis Using Terrain Following

A valid argument for incorporating pitch data in the terrain load analysis concerns the use of the terrain load lookup table for flight vehicles such as a helicopter. For these vehicles, the pitch is often much more pronounced than ground vehicles, and may be significant in estimating terrain load. For example, a helicopter pitched down may see an entire city and a large processing load. A helicopter pitched up would see nothing but sky and a rather low processing load. Without considering pitch of the viewpoint, the processing load is taken as if the helicopter is looking straight ahead, which in this case may be a poor estimation. One inherent problem with incorporating pitch into terrain load analysis is that this new dimension causes a significant increase in size of the lookup table. Since, for flight vehicles, the pitch of the vehicle is independent of the pitch of the terrain, a large range of pitch values is possible and must be accounted for in the lookup table. In addition, it is difficult to predict the future pitch of a flight vehicle. A pilot can change the vehicles pitch rapidly, rendering the predicted pitch/load inaccurate. The best alternative approach is to use worst case pitch/load. Also, the other inherent problem with flight vehicles is that they do not terrain follow, and thus may assume any height above terrain. As with pitch, incorporating a Z value into the lookup table would add significant data. Thus, while the terrain load analysis approach utilized in this study is sufficient for ground vehicles, it is evident that additional factors need to be considered for flight vehicles.

6. COMMON MISSION FUNCTIONS

Simulations often require database manipulations above and beyond those required to generate an image. These computations typically involve target location determination or collisions between objects. Typical mission functions are vector ranging (or line of sight), terrain following (or height above terrain), projectile, and collision detection. Vector ranging determines the range or distance to a database face along a vector. Terrain following is used to determine the height of a position above the database. Projectiles and collision detection both involve intersections between vectors and database faces.

Special algorithms may be required to implement these functions, and dissimilar results on different platforms may occur. Also, the fact that an IG is involved in the computations makes scene load management more difficult since image processing time is lost to mission function computations.

6.1 Common Mission Functions Methodology

It is one goal of the Common Mission Functions (CMF) to eliminate dissimilar results because of different mission function algorithm implementations. The CMF is intended to run on the host simulator and will thus off-load the mission function computations from the IG. As previously mentioned, this will ease the scene load management task.

The CMF has a layered design. The interface to the host does not affect the computations produced by the CMF. Furthermore, a layer is associated with the database so that changes to the database representation are kept independent of the intersection computations.

The top-most layer is the application layer. This layer is changed to conform to a host software design. The middle layer is the Mission Function Module (MFM). The interface between it and the application layer is fixed, and it is the application layer that must be modified to conform to the MFM. The bottom-most layer is the Database Interface (DBI). It separates the database manipulation from the MFM so that the database representation can be changed without altering the functionality of the MFM, and thus the application.

6.2 Application Layer

The application layer is the host simulator's software. This software must meet an interface requirement in order to communicate with the MFM layer. Some flexibility has been incorporated into the design of the MFM so that changes to it may be avoided. The first feature is that the representation of an entity (or moving model) identifier is defined by the application level. In a DIS environment, the entity identifier consists of host, application, and entity IDs. The second feature is that the application can provide an interface to an IG if an IG can be used to execute some mission

functions. This may not be the case for a host working with a CSM. Finally, the interface defines the function calls to execute the mission functions and return the results.

Two applications were written to illustrate the flexibility that exists. The first is a test driver program. The entity identifier is simply an integer and an interface to an IG is simulated. Each of the features provided by the MFM is tested on a stand-alone workstation. The second application is the reconfigurable host simulator. The entity identifier is the DIS identifier previously mentioned. An interface to an IG is provided but does not imply a requirement for its usage.

6.3 Mission Function Module Layer

The MFM is responsible for bookkeeping and computing mission functions. The possible mission functions are collision detection, projectile, terrain following, and vector ranging. All mission functions consist of one or more vectors to be intersected with the database. A general task for the MFM is to convert each mission function to a set of vectors to be passed to the DBI. Then the returned results are interpreted according to the mission function type.

The Vector Ranging mission function requires a single vector to be defined. This vector is infinite in the positive direction. If the vector intersects with a database face then the distance to the face is the vector's range.

The Projectile mission function requires a finite length vector to be defined. If the vector intersects with a database face then the projectile is typically considered to hit the face. The vector is typically moved until a hit occurs. The movement occurs by updating one end point of the vector. The other end point is the end point updated during the previous field. This makes the projectile follow an approximate curve. The projectile mission function can be associated with a (projectile) moving model. In this case the projectile end points follow the model position updates.

The Terrain Following mission function consists of one or more vectors which extend from the center of gravity of a model. The end points of these vectors are the origins of vectors that extend infinitely in the z direction (both positive and negative). The distance to a database face for each vector is the model's height above (or below) the terrain at that point.

The Collision Detection mission function can have several definitions. One such definition is Offset Vector Collision Detection. This consists of one or more finite length vectors (called the offset vectors) which extend from the model's center of gravity. Each of these vectors has another vector (called the stick vector) extending from the offset vector's end point. If a stick vector intersects a database face then the model is considered to have collided with the database.

Mission function vector sets require the vectors to be transformed according to the model's attitude and position. A projectile vector is only transformed by position. The transformed vectors are then passed to the DBI for intersection calculations. Results are returned to the MFM by the DBI.

Each mission function result consists of a 2D and a 3D component. 2D components involve intersections with the database. 3D components involve intersections with models and possibly other terrain features. A result can have no components for either or both of the 2D and 3D types. Each vector in the mission function definition can have its own 2D/3D components. In a case where a vector intersects multiple faces of the same type, the closest result is returned.

The MFM has several modes of operation. Mission functions can be executed on demand by executing a specific mission function when desired. Mission functions can be grouped in subcycles (explained below) and executed in an iterative fashion. Load management can also be applied to this subcycling method. As previously mentioned, an IG can be used to execute some mission functions. Different modes are provided to assign all, some, or none of the mission functions to the IG.

Mission functions can be subcycled to improve the performance of the MFM. Subcycling assigns a frequency of execution to each mission function. A mission function can be executed once, every field, or every "nth" field. Some mission functions do not need to be executed every field, so subcycling can lessen the amount of work the host has to do each field.

Applying load management to this subcycling can result in some mission functions not being computed during a field due to time constraints. The aborted missions are assigned to the next subcycle. Ownship missions (if any) are given priority over remote ship missions. More frequent missions are higher in priority than less frequent missions when executed during the same field.

6.4 Database Interface Layer

The database interface portion of the common mission functions segment is the portion of the segment which accesses the database and performs the vector mathematics to determine database face intersections. At this level, the software does not know what type of mission function is being performed, it does nothing more than report intersection points and database face information back to the calling function. Figure 25 shows the block diagram design for the common mission function package.

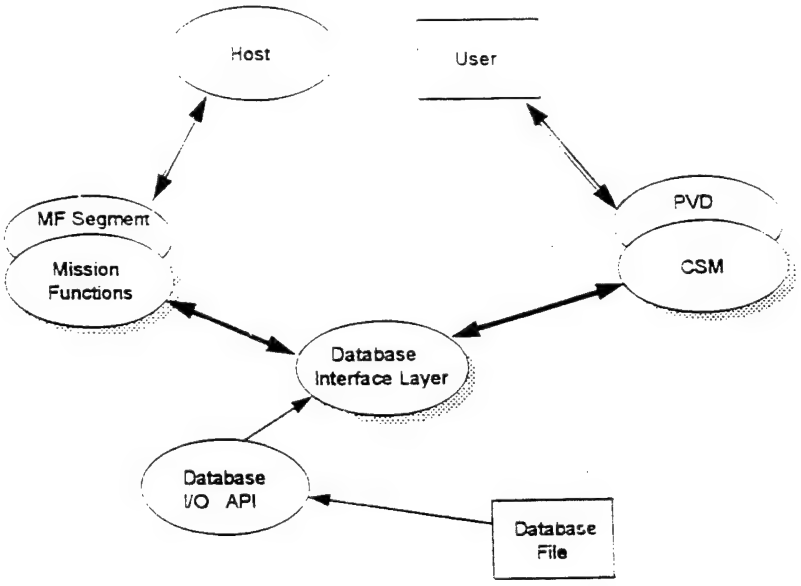


Figure 25 Mission Function Data Flow

To obtain the database information necessary for intersection calculations, this routine requires an API to exist which interprets the database format and loads it into a format compatible to its calculations. This API must be written for each database format that this routine is to interface with. It reads the database file and reduces the data within the file to polygons, edges, and points. Database information such as translucency, texture, or color are not necessary for intersection calculations and therefore are not carried with the CMF algorithm. The API needs to assign a region and cluster to each face if it does not already exist to assist in culling calculations. Furthermore, the API will calculate information not normally stored in database files which assists in speeding up intersection calculations such as edge plane normals (these are normals within the face plane that lie perpendicular to each edge).

Once the database has been loaded the database interface layer awaits for an intersection request. Upon this request, the DBI will begin culling out any large areas within the database which are unlikely to contain an intersecting face. Figure 26 shows the algorithm flow for finding an intersection with a database face. The hierarchy of data elements within the mission function database is regions, clusters and then faces as shown in Figure 27. The search for the intersecting face begins with a search within the current cluster. If a face cannot be found within the current cluster, then the current region is checked to obtain an adjacent cluster to search. If an adjacent cluster within the same region contains no faces which intersect with the vector, then the entire database is search by gathering regions which may intersect with the vector. Then the process continues with clusters and faces.

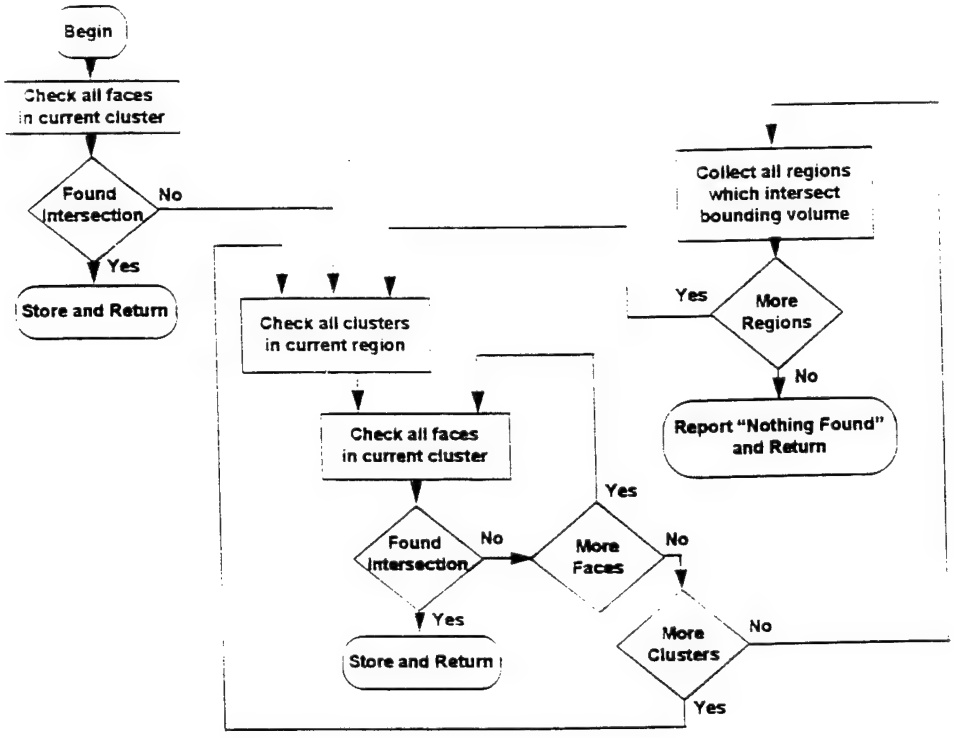


Figure 26 Culling Algorithm Flow

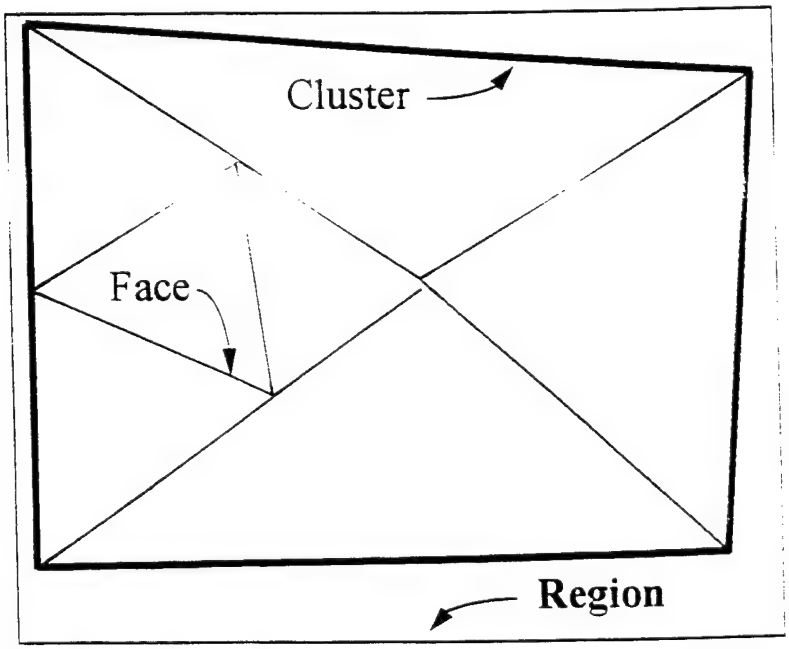


Figure 27 Database Culling Hierarchy

The culling process is achieved by a culling volume around each object in question. For example, a region has a volume associated with it defined by a radius. If the vector in question cuts through this volume, the region must be investigated for cluster and face intersection. The same holds true for clusters, that is, each one has a culling volume associated with it. The criteria which must be met to be included within the search list is the perpendicular distance from the vector to the center of the region/cluster must be less than the objects culling volume radius. The equation for the perpendicular distance from the vector V to the object with centroid C is:

$$D = C - \frac{C \cdot V}{V^2} V$$

Once the culling process is complete, the resultant face list is searched for intersections with the vector. The intersection with the smallest distance to the origin of the vector is kept and passed to the mission functions segment. The equation for the distance K along the vector V from the vector origin V_o to a plane with normal N_p and distance to origin d is:

$$K = \frac{-d - (\vec{V_o} \cdot \vec{N_p})}{\vec{V} \cdot \vec{N_p}}$$

7. TESTING / INVESTIGATIONS

Testing, investigations and evaluations were performed throughout the contract in three main areas. The CSM algorithm was tested and evaluated through the use of several scenarios. An investigation was performed to understand mechanisms for integrating multiple CSMs within the same network on the same exercise. The resultant design was tested for validation and verification through small single simulator scenarios and then for feasibility studies using multiple simulators in a large exercise. An investigation into interoperability employing continuous terrain was also performed on this contract. This study explored manipulating continuous terrain to overcome correlation problem between simulators. Each of these investigations will be discussed in the following sections.

7.1 Multiple CSM Study

As DIS environments become large, the CSM processing load can exceed the capacity of a single CSM. A multiple set of CSMs can be used to divide the load. This section describes a geographic approach for using multiple CSMs.

7.1.1 Design

In a multiple CSM system, each CSM is assigned a set of simulators to monitor and control via LOD datum records in Set Data PDUs. Because simulator ownship vehicles can navigate to places remote from starting positions, it makes sense to consider use of separated geographic areas of control for each CSM. Thus, the set of simulators observed by a CSM may change during an exercise. A major part of this study was to develop a means for one CSM to hand off control of a Simulator to another CSM. The design methodology for accomplishing this transfer is explained in this section. Figure 28 illustrates the geographic conditions of a multiple CSM exercise.

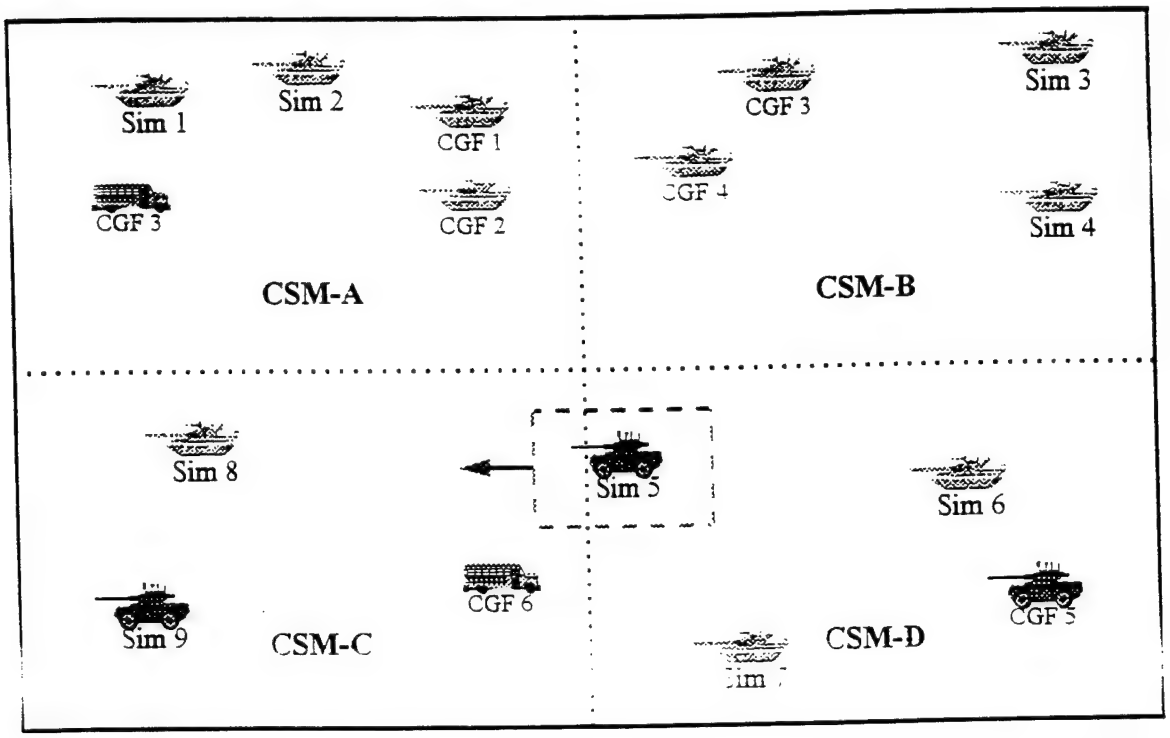


Figure 28 Battlefield Divided Into Geographic Areas Monitored by CSMs

Simulator "Sim 5" is about to leave the geographic area of CSM-D and enter that of CSM-C. CSM-D will detect this event and request that CSM-C take control of Sim 5. CSM-D will provide CSM-C with important information about the simulator. CSM-C will accept the hand-off, seek information from the simulator itself, and begin performing CSM algorithms for that simulator. CSM-D will no longer concern itself with the performance of Sim 5.

The subsections that follow present the algorithms used for a geographic based, multiple CSM architecture. The order of algorithms follows a chronological order of events occurring during the hand-off of a simulator from one CSM to another.

7.1.1.1 CSM Initialization Database File

In the original single CSM case, at initialization, a file, "sim_niu.list" was read. It contained, for each simulator to be monitored, the site number, the application number (also called the host number), a channel number, and the processing cycle rate of the simulator's image generator. In the new, multiple CSM case, each CSM reads an expanded "sim_niu.list". This file contains information about each CSM: site and application numbers, number of simulators, and rectangular boundaries. Actual simulator data appears in the same format as in the single CSM case. Figure 29 shows a sample file's contents.

```
# FILE: /proj/dis/baseline/tables/sim_niu.list
#
# number of terrain grid load ("load.dat") files
4
# load.dat path names
#
/proj/dis/user/tackaber/objlib/csm/src/ivacc_load.dat.airport
/proj/dis/user/tackaber/objlib/csm/src/ivacc_load.dat.iitsec
/proj/dis/user/tackaber/objlib/csm/src/load.dat.discr_airpt_fine
/proj/dis/user/tackaber/objlib/csm/src/load.dat.contn_airpt_fine
#
# number of CSMs
2
# CSM A: sun17
#     site, application, number of simulators monitorerd
#     Geographic Bounds (min x, min y, max x, max y)
#     For each simulator: site, application, channel, IG Rate,
#                         index for load.dat path name
78 17 3
0.0 0.0 207476.0 400000.0
78 13 0 30 1
78 28 0 30 3
78 16 0 30 2
# CSM B: sunstar
78 24 1
207376.0 0.0 400000.0 400000.0
78 14 0 30 1
```

Figure 29 Sample CSM / Simulator NIU List

Using calls to the Exercise Manager's NIU (Network Interface Unit), each CSM can determine its own site and application numbers. It can then use the sim_niu.list file to extract its own boundary and simulator information. The CSM also creates a table of *remote* CSM information. It will use this information to determine which CSM it should hand off a simulator to.

7.1.1.2 Tracking Simulators for Possible Hand-off

In CSM's real-time processing loop, predictions (extrapolations) of position are made for each simulator's ownship vehicle. The CSM compares each position with its rectangular boundaries defined in file, sim_niu.list. If it finds that the position does fall outside the boundaries, CSM next checks the boundaries of remote CSMs to find which CSM should take over control of the simulator which represents that vehicle. The CSM then initiates the hand-off process.

It should be noted that adjacent CSM geographic regions should have a small area of overlap. The overlap regions serve as deadband zones which prevent simulators from being continually passed back and forth between two CSMs.

7.1.1.3 Set Data PDU for Simulator Hand-off

The mechanism for communicating a request for hand-off of a simulator to a CSM is the Set Data PDU. [The proposed "Entity Handover" PDU was considered, but not chosen because it is limited to transferring an entity, whereas the CSMs are dealing with *simulators* that control entities.] The Set Data PDU contains a variable datum record that provides information that the target CSM will need to monitor the new simulator. The format of this datum record is shown in Figure 18.

When a CSM receives this Set Data PDU, it must decide if it can handle the requested assignment. If it can, it will send a Data PDU, containing the same datum record, back to the requesting CSM. If it does not want the added task, it will send an "empty" Data PDU back to the requesting CSM. The empty Data PDU contains no datum record, but does contain a request ID that matches that of the original Set Data PDU.

7.1.1.4 A CSM Takes Control of a New Simulator

When a CSM agrees to monitor / control a simulator formerly monitored / controlled by another CSM, it needs to bond with the simulator and its ownship. The hand-off datum record provides only minimal information. To find out more about the simulator, the CSM sends it a Data Query PDU asking for IG properties, terrain properties and moving model properties. The format of this PDU is identical to that used by the other CSM when it first received an Entity State PDU from that simulator (see Figure 9). Figures 10 through 13 show the format of the Data PDU that the simulator sends back to the querying CSM.

The CSM constructs a class object (CsmViewPoint) for the simulator and adds it to its linked list of simulators being monitored. A default Data PDU with IG, terrain, and moving model properties is also created. This will be overwritten by the Data PDU received by the simulator which was discussed previously.

Given the new ownship entity ID (from the hand-off datum record), the CSM can find the corresponding DisEntity class object address which has been maintained by the CSM's Exercise Manager NIU since the entity was first detected (well before the CSM hand-off). This object contains the address of the ownship's Entity State PDU data which is continually updated over the DIS network. With this information, the CSM can track the entity.

For all remote entities known by the Exercise Manager, CsmEntity class objects are created and kept in a linked list. These entities can then be displayed in scrollable lists which are used for model priority modification or for choosing an entity to be the target location for angular LOD control.

The CSM also maintains class objects (MemberCSM) about itself and other CSMs based on data from file, sim_niu.list (see Figure 29). The CSM adds the new simulator record to its own object and removes the simulator record from the object for the CSM which handed off the simulator.

Finally, the CSM redisplays the simulator table on the main CSM GUI window. The new simulator shows up on this list.

In summary, this subsection has listed the tasks that a CSM must perform in order to achieve the same operational state attained by the original CSM in regards to gaining access to a simulator to be monitored and controlled.

7.1.1.5 A CSM Relinquishes Control of a Simulator

The CSM that gives up control of a simulator must disassociate itself from that simulator. When the CSM gets the *acknowledging* Data PDU from the CSM to which it handed off the simulator, it searches its list of (CsmViewPoint) simulator objects and compares Site/Application IDs with that of the hand-off datum record. Finding a match, the CSM deletes entity objects associated with the simulator before deleting the simulator object itself.

The CSM removes the simulator record from its own MemberCSM object and adds the simulator record to the remote CSM now in charge of that simulator.

7.1.2 Evaluation

In a Multiple CSM test scenario, Exercise Managers are run on two workstations. One contains CSM A and the other contains CSM B. Each CSM is assigned a separate list of simulator hosts. During the scenario one simulator ownship, for example, the reconfigurable helicopter, traveled from CSM A's domain into CSM B's domain. Various controls were applied to the helicopter simulator from each workstation during the time that each CSM had control of the simulator. Simulator data displayed on the main CSM windows were observed for each CSM. At the time of hand-off, the helicopter simulator record disappeared from CSM A's display and appeared for the first time on the CSM B's display.

7.1.3 Results/Conclusions

The test scenario shows that the geographic based multiple CSM system can be utilized to accommodate many CSMs within a single exercise. The reconfigurable helicopter started out under control of CSM A. Continuous terrain was modified by CSM A and the results of the modification was a change in the mountain shapes on the simulator display. As the helicopter traveled from site 1 of the scenario to site 2, it passed into the geographic space controlled by CSM B. At that point a hand-off from CSM A to CSM B took place. The simulator lists on the CSM GUI's changed to show this hand-off. On arrival at site 2, CSM B commands were issued to control model LODs for the trucks that occupied this site. The changes in LOD were observed in the helicopter display.

7.1.3.1 A Performance Based Multiple CSM System

As part of a future project dealing with CSM, it might be worthwhile to consider using a "performance based" multiple CSM system. During an exercise, it is possible that some CSMs may have to work harder than others due to the distribution of simulators and remote entities, especially when there are a large number of entities controlled by Computer Generated Forces. If a CSM were made to monitor its own processing load and to periodically communicate this information to the other CSMs, it could be determined when busy CSMs should hand off some of their work to other CSMs carrying lighter loads. This might be preferable to increasing the processing cycle times of busy CSMs.

7.2 Interoperability By Continuous Terrain

The interoperability by continuous terrain study evolved through several designs, which, for completeness, will all be discussed in the following sections. Prior to the design, however, a discussion on the goal of this study is in order.

The goal of this study is to determine the merit and feasibility of overcoming interoperability issues by adjusting the level of detail in continuous terrain. Continuous terrain offers the unique characteristic of adjusting vertex positions, i.e. elevation, in a continuous manner. This characteristic of terrain might allow some types of interoperability problems to be compensated for by adjusting the continuous terrain elevation. This method of correction would allow a "fair fight" environment while keeping the detail required to meet a fair fight to a minimum.

Detection of interoperability issues is outside the scope of the CSM. If an interoperability server were devised however, the CSM could work in conjunction with this server to dynamically adjust terrain LODs to overcome specific interoperability issues. Figure 30 illustrates the mechanics of operation between an intervisibility server and the CSM. If the intervisibility server determined that terrain correlation between simulators was significantly low, it could inform the CSM that the terrain LOD

needs to be adjusted to compensate for this difference. The CSM could then make the decision to raise or lower the priority of terrain LOD modifications based on this information. The CSM already has the ability to adjust LOD of terrain for both continuous and discrete types of terrain. The only required enhancements to the CSM for this function would be to provide the means by which the intervisibility server could inform the CSM to adjust the LOD and then by how much. Since the intervisibility server did not exist for this study, these mechanisms for information transfer were not developed.

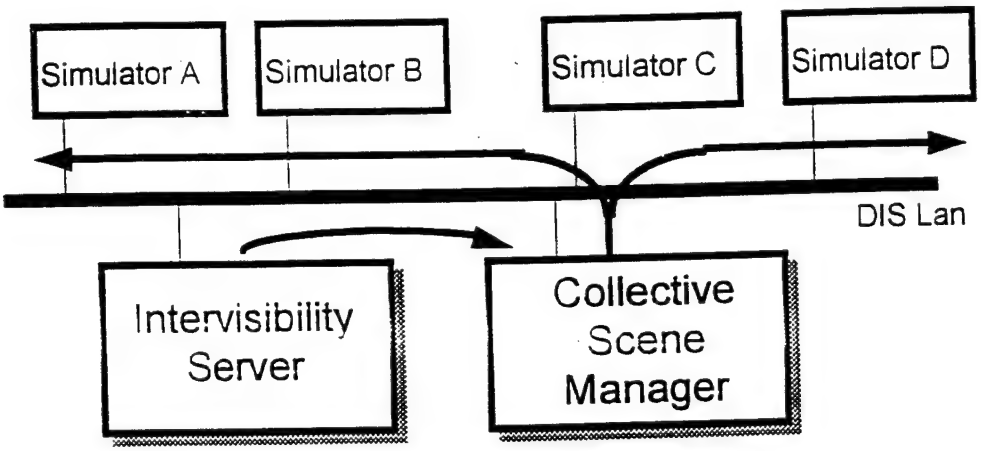


Figure 30 Interoperability By Continuous Terrain Design

7.2.1 Design

The original design called for locating a common geographical area in a discrete terrain form database and in a continuous terrain form database. The geographic area chosen for this study was Alaska since a database in both terrain forms existed for this area. These databases were developed under the IVACC contract and met all the database requirements without further development on this BAA. The next requirement was to find regions in these databases which presented terrain differences or areas of low correlation. Once these areas were found, the design called for applying several mechanisms to measure and asses feasibility in both a quantitative and qualitative manners. These mechanisms were as follows:

1) The line of site tools within the CSM/PVD would assist in detecting the areas of study as well as assisting in visualizing the solutions. If the line of site calculations determine there is an interoperability problem, and adjusting continuous terrain solves the problem, the line of site tools will then show that the problem has been rectified. In this regard, the line of site tool would act as an intervisibility server.

2) The AAR associated with the Exercise Manager would be used to log confrontation situations and determine when engagements occur and what the outcome was. If interoperability problems exist, one simulator would produce either no engagement or a missed firing while the other would indicate a full

engagement was in process. If the same situation occurred under correlated terrain conditions, both simulators would engage or hit the other target and a "fair fight" would take place. These actions can be logged into an after action review and used to illustrate the effectiveness of the terrain LOD adjustment on correlating databases.

3) Observing differences in terrain elevations between the two forms of terrain would quantitatively allow us to measure the interoperability problem as well as the effectiveness of the solution. The differences in terrain would be obtained by getting the elevation at a specified point in the discrete terrain database and then the corresponding elevation of that same point in the continuous database and subtracting the two values. Several points within a small geographic area could be collected to create a sampling or a grid of terrain elevation differences. It necessary to gather this sample of elevation points some distance from the viewpoint so that LOD adjustment can compensate for the differences. Care must be taken, however, in this collection, to insure that the distance from the viewpoint doesn't transition a discrete terrain LOD thereby introducing further terrain differences.

Initial investigations into terrain differences showed that both versions of the IVACC Alaskan databases were highly correlated. Measurement 3 above, was used to determine correlation by taking the differences in terrain as a sampled grid across the database. The maximum elevation difference between the discrete terrain and continuous terrain was 0.138 feet at the highest LOD in a mountainous area. Locating an area in both databases that was in error of each other was not possible using these databases. These databases, however, were the only databases which existed both forms of terrain and covered the same the same geographic areas.

Since uncorrelated areas in the Alaskan databases were not found, the study would have to degrade correlation between the two databases. This could be accomplished by adjusting the discrete terrain to its lowest level of detail, thereby introducing correlation errors with the higher level of detail continuous terrain. Continuous terrain would be adjusted to more closely resemble this lower level of detail discrete terrain and thereby better correlating the databases.

With this new approach, the tools which test and measure correlation could no longer be used as defined in the original approach. The PVD tools and host mission functions only load the highest level of detail terrain to work with. Since one representation of terrain on the image generator is at its lowest level of detail, the measurements which these tools provide would be uncorrelated with the visual system. An alternative approach was to use the image generator to measure terrain and locate differences.

As the study progressed, it was further found that the IG did not produce valid terrain elevations for continuous terrain in lower LODs. Continuous terrain, at the time of this study, was a new feature on the SE line of image generators and apparently not fully implemented in terms of mission function results. Without the ability to collect terrain elevation information on continuous terrain at lower levels of detail, quantitative measurements could not be obtained. Visual correlation was the only assessment

available to the study. With this in mind, a scenario was defined which allowed easy visualization of correlation when the continuous terrain LOD was adjusted.

7.2.2 Evaluation

An area of the database was found which presented extremely rough terrain and could easily be made different by adjusting the LOD of the discrete terrain. Two helicopters were strategically placed in this area on opposite sides of a sharp peak at a mountain crest. The viewpoint was positioned in perspective mode (100 feet behind and 25 feet above) behind one of the helicopters. When the LOD of discrete terrain was adjusted such that peak was lowered both helicopters were visible in the discrete version. The remote helicopter was still masked, however, in the continuous terrain version.

An obvious visual correlation problem now existed between databases. If LOD differences had not been the cause of this miscorrelation problem, the line of site tools could have been used to easily detect the interoperability problem.

The measured discrete terrain elevation difference from highest LOD to this lower LOD was 42 feet. The mountain top then, was lowered by 42 feet. The method of adjusting the continuous terrain LOD is an error per range mechanism and if this was adjusted to 150 from its nominal 19 value, then the mountain top lowered in the continuous terrain database and the remote helicopter also became completely visible in this simulator.

The continuous terrain has many intermediate LOD representations between the initial higher resolution and the discrete-matching lower resolution. For example, at an error per range of 110, the helicopter was only partially visible. This indicates that the continuous terrain could achieve closer correlation to the discrete database at any level of detail between the highest and lowest representations.

7.2.3 Results/Conclusions

Several conclusions can be drawn from this study. The first and foremost is that continuous terrain can be adjusted to compensate, at least to some degree, for terrain miscorrelation. The most difficult area of the design is that of automated detection of correlation problems. The intervisibility server which detects correlation problems, in the form of terrain elevation differences, and informs the CSM becomes an extremely difficult problem to solve. Line of site interoperability problems are easy enough to detect, but the server would have to determine that the problem was caused by terrain elevation differences rather than culture differences or moving model placement. This is an extremely difficult problem. To further complicate the role of the intervisibility server, if it determined that terrain was miscorrelated, it must then determine by how much.

The study also indicated that methods of incorporating LOD information into the tools and host mission functions should be done. Terrain in an image generator is a

dynamic environment which changes LOD constantly as the viewpoint changes. If a tool or mission function is to perform line of site using the terrain in manner which visual correlates with the scene being rendered on the image generator, it must take into account the terrain LOD. This is not an easy problem however, since there are as many methods of adjusting LOD as there are simulators.

7.3 CSM Feasibility Study

A demonstration which assesses the plausibility of the operation of the collective scene manager in a "hands-off" manner was also performed under this phase. By "hands-off" manner, it is meant that an IG will be in a true overload situation and the CSM must predict the overload occurrence and compensate for it without the user manually invoking a CSM control. This was the first chance the CSM had to operate in a true DIS environment and to control more the a single IG at one time. To assess feasibility of the design, a set of requirements must first be defined. These requirements will be briefly discussed here.

The requirements must illustrate the capabilities of the CSM to date. The CSM must be allowed to react, without user interaction, to image generator overload by detecting potential overload within the scene based on entity transitions within the exercise. To allow the CSM to predict the overload situation, the targets are required to converge into a single field of view or area. To successfully avoid image generator overload, the CSM should reduce both moving model Level Of Detail (LOD) and terrain LOD to manage the overload situation. Standard forced LOD control should be demonstrated as well and angular LOD. The CSM is required to control scene load on two IGs employing different terrain type databases, i.e. continuous and discrete. Each IG will be controlled by an independent host.

7.3.1 Design

The requirements listed above drove the scenario design which would demonstrate the CSM's capabilities. A scenario which met all the requirements was designed for the Alaskan database and it incorporated two simulators with two host controllers.

The scenario utilized the Alaskan database because it existed in both discrete terrain and continuous terrain formats. The exercise was located at the large airport in the Southwest area of the Alaskan database. As shown in Figure 31, from the south 8 foe vehicles approach the airport. Three are T-72 tanks and five are ZIL-157 trucks. From the east two friendly AH-64 helicopters approach. Two AH-64 helicopters also approach from the west. This totals 12 vehicles within the exercise converging upon the center of the airport. The viewpoint could be attached to any of these vehicles. Ground vehicles however provided the best opportunity to see terrain LOD changes in the background with target LOD changes in the foreground, so the viewpoint was attached to two friendly M1A1 tanks on the southwest corner of the airport.

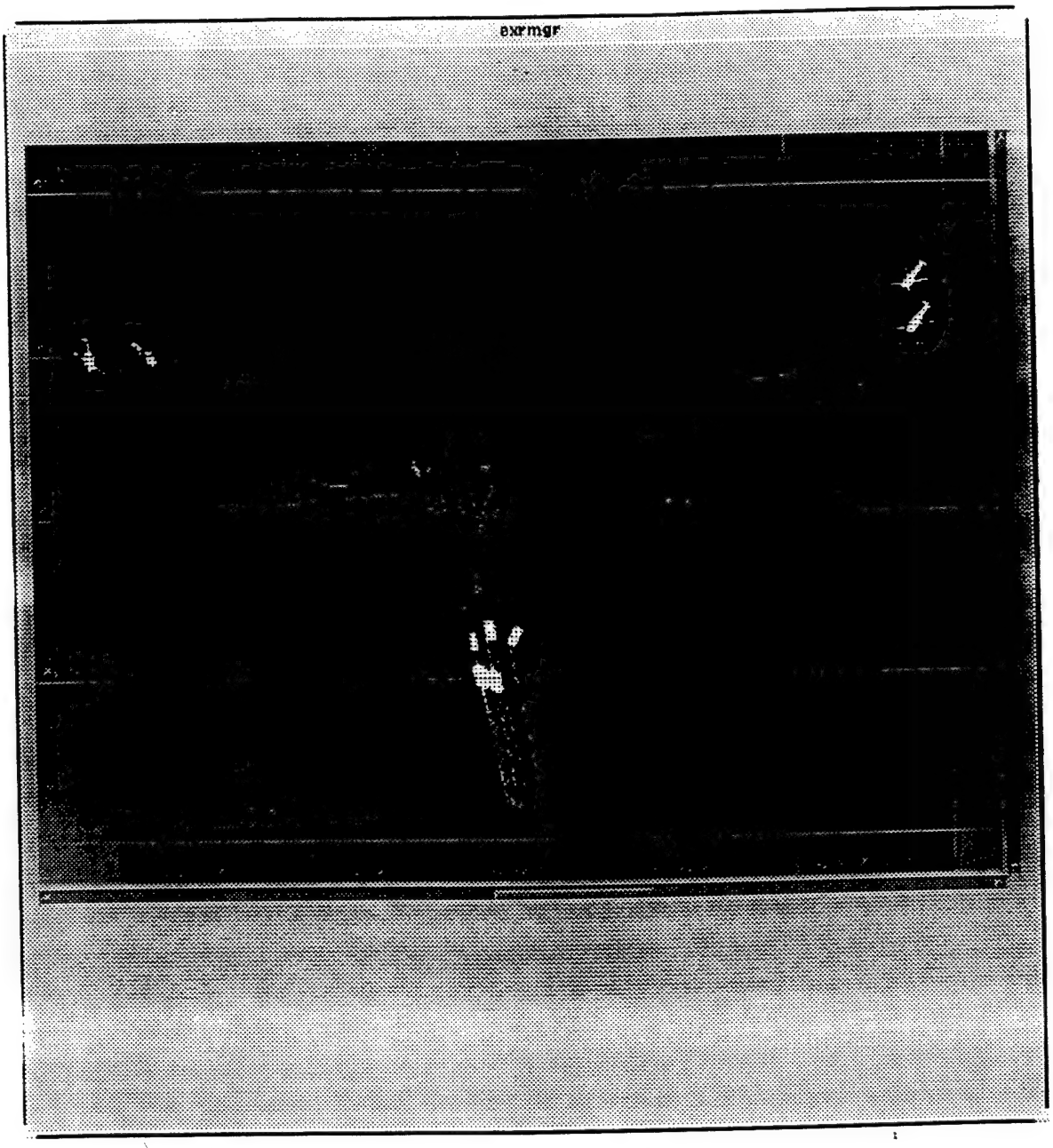


Figure 31 CSM Demonstration Scenario

As the exercise began. the tanks invaded the airport preceding the truck convoy. The truck convoy had an asset that the friendly forces wished to acquire so the task is to immobilize the tanks and take control of the convoy. The AH-64 helicopters begin to close on the airport a few seconds into the exercise. The helicopters engage the tanks and destroy them. Since the vehicles are all within the same local area. the CSM could predict a potential overload situation and the LOD of the moving models and the terrain could be modified accordingly.

Priorities, by LOD, were chosen for moving models and terrain so that during a period of overload prevention, CSM would first simplify the least important objects and try to maintain a high fidelity for important objects. In general, friendly vehicles and non-threatening enemy vehicles are given lowest priority. Vehicles that pose a threat to the ownship are given a high priority. The importance of high fidelity terrain might depend on the particular mission. In the scenario, terrain was given an intermediate priority. A complete list of priorities used in the scenario is given in Table 2. In this table, the priority values are an indication of importance. The higher these values are, the more important the object is to the exercise. The M1A1 tanks to which the viewpoint was attached are not listed in the priority table because they were simply a stealth vehicle in the exercise and not participants. The terrain percentages listed in the table refer to the range ring mechanism with which terrain is controlled. In this manner 100% to 85% refers to the range ring distance being between 100% and 85% of its actual distance for that range.

Table 2 Terrain / Moving Model LOD Priorities

Terrain / Model	LOD Transition	Priority
ZIL Truck	LOD 0 to LOD 1	10
ZIL Truck	LOD 1 to LOD 2	10
AH64	LOD 0 to LOD 1	20
Terrain	100% to 85 %	40
AH64	LOD 1 to LOD 2	60
Terrain	85% to 70%	80
T72	LOD 0 to LOD 1	100
ZIL Truck	LOD 2 to LOD 3	110
T72	LOD 1 to LOD 2	1000
AH64	LOD 2 to LOD 3	2000
T72	LOD 2 to LOD 3	10000

Normally, one would expect a truck LOD change to be the first CSM event observed during an overload correction. However, there are two reasons why this might not occur. First, some low priority moving models may undergo LOD changes outside the field of view and hence, not be observed. Recall that load management is done per angular sector around the viewpoint because the viewpoint's future (e.g., 5 seconds from present) heading cannot be assumed. The second circumstance occurs when for some sector, the processing load of terrain by itself or with higher priority models is near or at overload. In this case a correction is first made for terrain LOD. And, since terrain changes are universal, the effect is observed by the viewer independent of which sector has the process load problem.

In order to achieve an actual overload situation on the image generators (two SE1000 IGs), each IG drove 4 separate channels. The demonstration only displayed one per simulator but the reconfigurable helicopter displays were available to see the other channel displays. Terrain load information was gathered for the airport areas of the database for both the continuous terrain and the discrete terrain databases in the four channel IG configurations.

Once this design and scenario were complete, the exercise was executed and logged using MAK Technologies logger and this log file drove the demonstration and was used to assess CSM feasibility.

7.3.2 Evaluation

The exercise described in the previous section was executed and CSM performance was assessed. To verify that the an overload situation was induced by the scenario, the exercise was run first without the CSM. The discrete terrain IG scene began to break up with models, culture and terrain popping in and out violently. The continuous terrain IG overload was more subtle and only noticeable when motion occurred. The IG was operating at a lower frame rate to compensate for the overload situation so normally smooth joystick motion appeared jerky.

With the CSM added to the exercise, the overload was controlled and the scene was much more visually appealing. Prior to the vehicles entering the field of view, the mountainous region on the far east side of the airport began to show LOD adjustments in both discrete terrain and continuous terrain. When the set of tanks actually entered the field of view, the terrain LOD was again adjusted in an effort to bring back some detail. This happens when the CSM determines that the load was not as high as it could be and hence detail can be increased. Just prior to the trucks entering the field of view, the terrain was once again adjust to a lower level of detail and this time remained there. The trucks entered the field of view and some of them had been adjusted to a lower level of detail. The scene was entirely stable and all elements necessary for a "fair fight" were still present.

An interesting observation which occurred during the feasibility study was that even though all the elements of the scene were visible in both IGs, some elements differed in LOD between the two simulators. The trucks, for example, were at lower LODs in the discrete terrain simulator than the same trucks rendered by the continuous terrain simulator. The explanation for this occurrence is that discrete terrain does not provide as much processing time relief as continuous terrain when lower LODs are used. This causes more of the moving models to be adjusted to compensate for the overload. It was observed that four trucks were at lower LODs in the discrete simulator compared to a single truck in the continuous terrain simulator. This behavior is the desired behavior of the CSM since it controls different simulators independently according to the capabilities of that simulator. The end result is a "fair fight" rendered scene without overload.

7.3.3 Conclusions

The CSM operated entirely as expected and according to design. It proved to be a viable mechanism to overcome overload situations in a distributed exercise in a correlated manner. For this particular demonstration, the CSM performed flawlessly and fully compensated for the overload situation.

A great deal of time was spent in preparation for the exercise from an integration standpoint. The exercise had to be designed and implemented using the Exercise Manager. The terrain load information had to be collected. Two IGs and two hosts had to be integrated and proper communication between them had to be established and verified. Priorities for the moving models and the terrain had to be determined. Once the scenario yielded an overload situation, it then had to be logged to preserve events. The time spent in this preparation was necessary for a successful demonstration, however, it would be diminished slightly, if the CSM did not require as much a-priori information about the simulators and the exercise.

8. SOFTWARE DELIVERY

8.1 CSM

The software which comprises the Collective Scene Manager is discussed in the following sections. These sections describe the specifications and integration design which are necessary when attempting to interface to the software.

8.1.1 Code Specification and Software Integration

This section contains lists of files used in making the Collective Scene Manager. It also contains short descriptions of C++ coded functions. These files are included in a tape delivered with this report.

8.1.1.1 CSM Library Files

CSM is a part of the Exercise Manager which, as an executable file, is made up by linking numerous compiled library files. The bulk of CSM code makes up one library that is linked into the Exercise Manager. The source and include files making up this library include the following:

csm.cc	csm.hh
csmApp.cc	csmApp.hh
csm_callbacks.cc	
csm_entity.cc	csm_entity.hh
csm_gui.cc	
csm_main.cc	csm_main.hh
csm_stubs.cc	
csm_viewpoint.cc	csm_viewpoint.hh
member_csm.cc	member_csm.hh
terrain_grid.cc	terrain_grid.hh
	csm_datums.h
	csm_externs.hh

The CSM graphical user interface is constructed using X-Designer developed by Imperial Software Technology. The construction process builds file, csm.xd and several of the files above (csm_gui.cc, csm_externs.hh). Also built is file stubs.cc, whose functions are copied to and expanded in file csm_stubs.cc listed above.

Files Makefile and defs.mk are used to *make* the library file. libcsm_sun_os5.a which gets linked in with other library files to create executable file. exrmgr. exrmgr is the Exercise Manager.

Another Exercise Manager library, libdis_view_sun_os5.a includes source files used for CSM's Plan View Display Tools. These files are:

EntitySectorLoadView.cc	EntitySectorLoadView.hh
EntityVectorLOSView.cc	EntityVectorLOSView.hh
EntityLOSView.cc	EntityLOSView.hh

8.1.1.2 CSM Support Files

The Exercise Manager reads in setup files at startup. Two such files are: "exrmgr.setup" and "niu.setup". These files contain path names, file names, and various parameter values. Network port numbers, site IDs and Host numbers are examples.

As discussed previously, each Collective Scene Manager reads a file. "sim_niu.list". This file contains information about each CSM. A sample file content is shown in Figure 29.

Terrain grid process load data is gathered by a the terrain load software program discussed in section 8.3. This program is run standalone and it communicates with the Image Generator. The load data is stored in files and is read later by CSM. The "load.dat" files generated during this phase are listed below.

ivacc_load.dat.airport	- data collected in region near airport
ivacc_load.dat.iitsec	- region used in I/ITSEC 95 scenario.
load.dat.discr_airpt_fine	- discrete database near airport
load.dat.contn_airpt_fine	- continuous database near airport

8.1.1.3 Code Function Description

Included in this sub-section is a list of functions found in CSM source code. A short description of each function is given.

Class Csm - csm.cc, csm.hh

FUNCTION	DESCRIPTION
AddModel	Add a moving model entity to all viewpoints.
Csm	Constructor. Initialize several variables.
DisplaySimList	Redisplay simulator info on main CSM Window.
ExerciseInProgress	Called when CSM comes on-line while Exercise already in progress. For each simulator we control, check for ownship and external entities. Bookkeep.
ForceRangeFactors	For each Viewpoint, force range factors for all entities.

GetUpdateCycleCount	Get update cycle count for CSM processing.
GetLine	Read text from file containing CSMs/Simulators.
GetNumViewpoints	Return number of viewpoints currently being monitored
GetViewPoint	Return a pointer to a viewpoint class object for a given viewpoint index.
HandoffAcknowledged	Receive acknowledge from another CSM that it is taking over control of a simulator. Bookkeep.
InitCsm	Initialize parameters. Call InitCsmViewPoints.
InitCsmViewPoints	Read List of CSMs and Simulators that can be monitored and controlled. Read and store grids of terrain process load data. Create MemberCSM objects with pointers to simulator data. Create CsmViewPoint objects for each simulator. Set default simulator information (to later be overwritten by actual host supplied data). Display the CSM window with Simulator list.
IGInit	For the Viewpoint having site and host matching those of a Data PDU, call InitModelMap to store IG information.
LoadManage	For each Viewpoint in viewpoint list: Call ComputeLoading function. Send PDUs with recommendations for change. Check for Hand-Off of ownship simulator to another CSM. If so, send request PDU.
MakeDefaultSimInfoPDU	Make a Data PDU with typical simulator properties, terrain properties, and moving model properties.
RemoveModel	Remove a moving model entity from all viewpoints.
SendDataHandoffAcknowledge	Send PDU to another CSM to acknowledge that we are taking control of a simulator.
SendSetDataHandoffRequest	Set up PDU to request hand-off of simulator by another CSM.
SetDebug	Toggle on/off Debug display for all viewpoints.
SetIGLoad	For the Viewpoint having site and host matching those of an IG Load PDU, store the IG load.
SetLookAheadTime (Get)	Set look-ahead time for all viewpoints.
SetNiuProps (Get)	Access to Network Interface Unit properties.
SetNumPriEntities (Get)	Set number of entities having priorities.
SetPriorityRecID (Get)	Set Entity ID of Moving Model for Priority.
SetPriorityRecWeight (Get)	Store priority weight of a moving model.
SetRangeClamp	Set Underload Range Clamp for all viewpoints.
SetRefreshFlag (Get)	Set flag for refreshing CSM window display.
SetTerrainPriWeight (Get)	Set Priority Weight for Terrain.
SetUpdateCycleTime (Get)	Set cycle time and count for CSM processing.
SimulatorHandoff	Receive PDU from another CSM requesting we take over control of a simulator. Bookkeep. Create CsmViewPoint object for simulator. Initialize. Request information from simulator. itself.

Class CsmViewPoint - csm_viewpoint.cc, csm_viewpoint.hh

FUNCTION	DESCRIPTION
AddEntity	Add a newly detected model to entity list. If it is first from a simulator, send data query to that simulator.
AddMissingEntities	Add new entities for a host after CSM comes back on-line and may have missed some activity.
AddPDUTerrainDatum	Add data for terrain control into Set Data PDU.
AddRemoteEntities	Add remote entities for a new monitored Host.
CoarserInsertSort	Sort entities in priority order (for overload).
ComputeInsideSector	Determine if entities are inside angular sector. If so, increment predicted process load.
ComputeLoading	Compute total load for IG: LookAhead for ownship and external models. Call UnderloadProcessing and OverloadProcessing. Call SendLODUpdates to realize changes.
CsmViewPoint	Constructor: Initialize variables to defaults.
~CsmViewPoint	Destructor
CurrentPredictedLoad	Compute predicted load of terrain and models for current ownship position and view.
DisplayPriWeightList	Display Priority Weight List for moving models.
dumpPDU	Debug dump of datum words for Set Data PDU.
ExternalLookAhead	Extrapolate external entity positions to look ahead time.
FinerInsertSort	Sort entities in priority order (for recovery).
ForceLOD	Force all entities to a specified LOD.
ForceRangeFactors	Restore range factors to unity for all entities.
GetAngularEnable	Determine if Angular LOD control is on.
GetAngularType	Retrieve type of Angular LOD control.
GetDataPDU	Get pointer to PDU with IG properties, etc.
GetEntity	Return pointer to entity (CSMEntity) object.
GetEntityID	Return Site/Host/Entity for moving model of given index.
GetForcedLOD	Get LOD to which all models have been forced.
GetFuturePos	Get extrapolated position of ownship.
GetHandoffInfo	Get information from hand-off PDU data.
GetHandOffInProgress	Simulator Hand-off Utility.
GetIGRate	Get cycle rate of Image Generator.
GetLoad	Return current IG processing Load.
GetLoadManageEnable	Determine if Load Management is active.
GetMaxFaceLoad	Return maximum process time threshold value.
GetMinFaceLoad	Return minimum process time threshold value.
GetNumEntities	Return number of moving model entities.
GetOwnshipPos	Compute local ownship position, given ES PDU.
GetRangeClamp	Return Enable Underload Range Clamp.
GetSetDataPDU	Return pointer to PDU to be sent to Host.
GetSetDataPtr	Return pointer to "data area" of PDU to Host.
GetSiteHost	Return site and host of simulator.

GetTerrainGrid	Return pointer to terrain grid table.
GetTerrainManageEnable	Determine if Terrain management is enabled.
GetTerrainPriority	Retrieve current priority of Terrain.
GetTimeLimit	Return time limit for one IG cycle.
InitModelMap	Extract IG properties, Terrain properties, and moving model properties from Host's Data PDU.
IsOwnship	Determine if ownship is active for simulator.
LoadFeedback	Not currently used. Compute viewpoint position and attitude. Compute range, process load, viewpoint azimuth for each moving model entity. Determine which entities are in the current viewpoint sector. Use mthd_lst_squares to compute a new process load threshold.
LookAhead	Extrapolate ownship position and attitude.
mthd_lst_squares	Not currently used. Use load estimates and stress factors as statistical feedback to compute new moving model load thresholds.
OverloadProcessing	For each angular sector of viewpoint Determine process load for entities in sector. Sort entities in order for making changes if beneficial, modify LOD transition range(s). Add in terrain processing time, if applicable. Repeat above until below face count threshold.
PrintEntityStatus	Debug Print of Entity load information.
RemoveEntitiesDueToHandoff	Remove entities associated with simulator no longer monitored.
RemoveEntity	Remove CSM entities when models are deleted.
SendAngularPDU	Build and send an Angular LOD control Set Data PDU.
SendDataQueryForIGLoad	Request host to send IG load periodically.
SendDataQueryPDU	Request information from newly discovered host.
SendLODUpdates	Send LOD update for all changed entities.
SendPDU	Send the Set Data PDU to the simulator host.
SetAngularEnable	Enable Angular LOD control.
SetAngularEntity	Operator has specified moving model to reference for angular LOD control.
SetAngularPower	Operator has specified cosine power for angular LOD control.
SetAngularPos	Operator has specified database reference for angular LOD control.
SetAngularType	Operator has specified type of LOD Angle control (boresight, moving model, database)
SetDebug	Toggle on/off Debug display.
SetFeedbackEnable	Enable or Disable Feedback Processing, where IG load data is used to influence changing of maximum face threshold.
SetForcedLOD	Set LOD value to be forced on all moving models.
SetHandOffInProgress	Simulator Hand-off Utility.
SetIGLoad	Store data from a host input IG load Data PDU.
SetLoadManageEnable	Enable or Disable Load Management
SetLookAheadTime	Set look-ahead time for predicting load and taking corrective action, if necessary.
SetMaxFaceLoad	Set max process time threshold to given value.

SetMinFaceLoad	Set min process time threshold. for recovery from previous overload correction, to given value.
SetPEntity	Set pointer to DIS Entity object for ownship.
SetRangeClamp	Set Underload Range Clamp for viewpoint.
SetSetDataPtr	Set pointer to data area of Set Data PDU.
SetTerrainManageEnable	Turn on/off terrain load manage enable flag.
SetTerrainPriority	Operator has modified priority of Terrain.
SetTimeLimit	Operator has set a new time limit to affect overload prevention and underload recovery.
SimulatorHandoff	Given new simulator ownship to monitor, construct and save information, entity list.
TerrainCoarserCosts	Compute cost benefit of bringing in Terrain Range Rings to next Scaling. Also determine where this terrain cost benefit fits in the Moving Model Entity (Cost Ordered) list.
TerrainFinerCosts	Compute cost penalty for moving out Terrain Range Rings to next Scaling. Also determine where this terrain cost penalty fits in the Moving Model Entity (Cost Ordered) list.
Test_1 ... Test_15	Test PDUs for Host are built and sent.
UnderloadProcessing	For each angular sector of viewpoint: Determine process load for entities in sector, Sort entities in order for making changes. If legal and beneficial, modify LOD range(s). Add in terrain processing time, if applicable.

Class CsmEntity - csm_entity.cc, csm_entity.hh

FUNCTION	DESCRIPTION
AddPDURecord	Add a range scale datum record to the Set Data PDU.
AddPDURangeScaleDatum	Formulate a datum record with range scale factor and transition time for the Host.
CoarseCompare	Compare costs for changing to coarser LOD for two entities.
CoarserAllowed	Return permission to change to coarser LOD.
ComputeCoarserLODCost	Compute the cost benefit of decreasing the LOD of the entity. Take into account: range scale factor and priority weights.
ComputeFinerLODCost	Compute the cost to increase the LOD of the entity. Take into account: range scale factor and priority weights.
ComputeInsideSector	Given the current azimuth angle of the entity from the viewpoint, and the angular limits of a sector, determine if model is inside sector.
ComputeRangeScale	Compute Range Scale Factor given entity range and default transition ranges.
CsmEntity	Constructor: Initialize parameters for entity. Call function SetModelLOD.
~CsmEntity	Destructor:
FineCompare	Compare costs for changing to finer LOD for two entities.
FinerAllowed	Return permission to change to finer LOD.
ForceRangeFactor	Set range scale for model.
GetCoarserLODCost	Return transition cost to move to coarser LOD.
GetCombinedPriority	Return Model Priority times LOD priority.
GetCurrentLOD	Return LOD being used in IG.
GetCurrentRange	Return range of model from viewpoint.
GetEntityID	Get Entity site, host and entity ID.
GetFaceLoad	Return model process Load at current range.
GetFinerLODCost	Return transition cost to move to finer LOD.
GetInsideSector	Return Yes/No for entity inside angular sector.
GetModelPriWeight	Return priority weight of an Entity.
GetPEntity	Return pointer to DisEntity object.
GetStartLOD	Return LOD being used at start of cycle.
GetTransitionCoarserFaceLoad	Compute face load difference for transitioning to the next coarser LOD from the current LOD.
GetTransitionFinerFaceLoad	Compute face load difference for transitioning to the next finer LOD from the current LOD.
LookAhead	Look ahead and return Face Count: Extrapolate entity position to future time. Compute azimuth angle of viewpoint looking at entity at this future time. Compute viewpoint to model range. Find LOD of model given range and range scale factor. Get process time for that LOD. Also compute costs for transitioning model to one LOD finer and one LOD coarser.

OutputPDU	Add a range scale datum record to the Set Data PDU.
PrintLODStatus	Print out entity information: Site/Host/Entity current LOD, load, range, IG range scale.
ResetLoadManage	Set start level of detail to current LOD.
ResetRangeScaleTo1	Return to normal LOD by setting range scale = 1.
SetModelLOD	Set Model LOD information (process time, default transition ranges, and priority weights) for each LOD. This information is found from the Host IG Data PDU.
SetModelPriWeight	Set priority for the moving model.
ThisLookAhead	Extrapolate position of moving model.
TransitionCoarser	Perform transition to coarser LOD, and compute the cost benefit in processing time count for doing so.
TransitionFiner	Perform transition to finer LOD, and compute the cost in increased processing time for doing so.
TransitionToLod	Perform transition to a specified LOD during a specified transition period. Calculate a range scale factor that will result in moving the transition ranges so that the entity will lie halfway between two ranges, and thus be stable at the new LOD. Set up range scale datum record for Set Data PDU to be broadcast to the Simulator of interest.

Class CsmSimulatorInfo - member_csm.cc, member_csm.hh

FUNCTION	DESCRIPTION
CsmSimulatorInfo	Constructor. Initialize parameters.
~CsmSimulatorInfo	Destructor.
GetHost	Return host or application number of simulator.
GetSite	Return site number of simulator.

Class MemberCSM - member_csm.cc, member_csm.hh

FUNCTION	DESCRIPTION
MemberCSM	Constructor. Initialize parameters.
~MemberCSM	Destructor.
AddSimulator	Add a Simulator Information object for the CSM member.
GetBoundary	Return database x and y minimum and maximum for CSM
GetHost	Return host or application number of CSM.
GetNumSimulators	Return number of simulators monitored by CSM.
GetSite	Return site number of CSM.
RemoveSimulator	Remove a simulator information object for the CSM member.

Class CsmApp - csmApp.cc, csmApp.hh

FUNCTION	DESCRIPTION
CsmApp	Constructor. Initialize View On/Off to Off.
~CsmApp	Destructor.
GetEntitySectorLoadView	Returns pointer to Entity Sector Load View.
GetEntitySectorsLoadIsOn	Tells if entity sector load is on or not.
GetLOSContourIsOn	Tells if Line of Sight Contour is on or not.
GetLOSContourView	Returns pointer to LOS Contour View.
GetLOSVectorIsOn	Tells if LOS Vector View is on or not.
GetLOSVectorView	Returns pointer to LOS Vector View.
GetTargetLODsIsOn	Tells if Target LOD view is on or not.
GetTerrainLoadGridIsOn	Tells if Terrain Load Grid is on or not.
SetEntitySectorLoadView	Set pointer to Entity Sector Load View.
SetEntitySectorsLoadIsOn	Set entity sector load view on or off.
SetLOSContourIsOn	Set LOS Contour view on or off.
SetLOSContourView	Set pointer to LOS Contour View.
SetLOSVectorIsOn	Set LOS Vector view on or off.
SetLOSVectorView	Set pointer to LOS Vector View.
SetTargetLODsIsOn	Set Target LOD view on or off.
SetTerrainLoadGridIsOn	Set Terrain Load Grid on or off.
Update	Based on View flags on/off, update views for current entities.

Files: cam_main.cc, csm_main.hh

FUNCTION	DESCRIPTION
CsmInit	Called from Exercise manager when CSM mode selected by operator. Set pointer to NIU properties. Call CSM object initialization function. Set up database origin. Make a CsmApp object. Call CSM object functions: ExerciseInProgress and DisplaySimList.
CsmLoop	Called from Exercise manager real-time processing loop. Call CSM function, LoadManage to direct further processing. Call csmApp function to Update PVD display.

Files: cam_callbacks.cc, csm_main.hh

FUNCTION	DESCRIPTION
CsmDataPDUReceivedCallback	Receives IG properties, IG process load, or acknowledge of simulator hand-off. Call appropriate CSM object function to process.
CsmEntityAddedCallback	Call CSM object function, AddModel.
CsmEntityRemovedCallback	Call CSM object function, RemoveModel.
CsmSetDataPDUReceivedCallback	Receives Set PDU form CSM for simulator hand-off. Acknowledges this PDU. Calls CSM object function SimulatorHandoff to process.
SimanPDUNotFromMyself	Is PDU trivially internal of from outside ?

Files: terrain_grid.cc, terrain_grid.hh

FUNCTION	DESCRIPTION
TerrainGrid	Constructor. Read process load data from input file and construct a 4-D grid of process load values. Dimensions on database row, col, angular sectors and LOD scale factors.
~TerrainGrid	Destructor. Free up allocated grid array.
GetGrid	Return pointer to 4-D grid of process load values.
GetNumAngleSectors	Return number of angular sectors.
GetValue	Compute bilinear interpolated process load values for a given database grid point, angular sector, and LOD scale factor.

8.1.2 Users Manual

This section contains a description of how to run CSM. It includes a brief discussion for running the Exercise Manager and details of the CSM graphical user interface.

8.1.2.1 The CSM Main Window

Since CSM is part of the Exercise Manager. the Exercise Manager must be run first. Describing the operation of the Exercise Manager is beyond the scope of this report, but a few details pertinent to CSM are included here. To run the Exercise Manager, set the current directory to one that contains an executable image, such as /proj/dis/baseline/build/exrmgr/sun_os5, and then type: ./exrmgr. When the Exercise Manager window appears, select "View (NIU)" from the mode menu, select "Load Database..." from the file menu, select a database, such as "itsec95", from the Load Database window's scroll list, and select the "Load Entire" button. After the 2-D database appears on the Exercise Manager Plan View Display (PVD), select "CSM

(NIU)" from the mode menu. At this point the Collective Scene Manager main widow appears. For a CSM assigned to monitor three simulators with Host IDs 13. 28. and 16, the CSM window will first appear as shown in Figure 32.

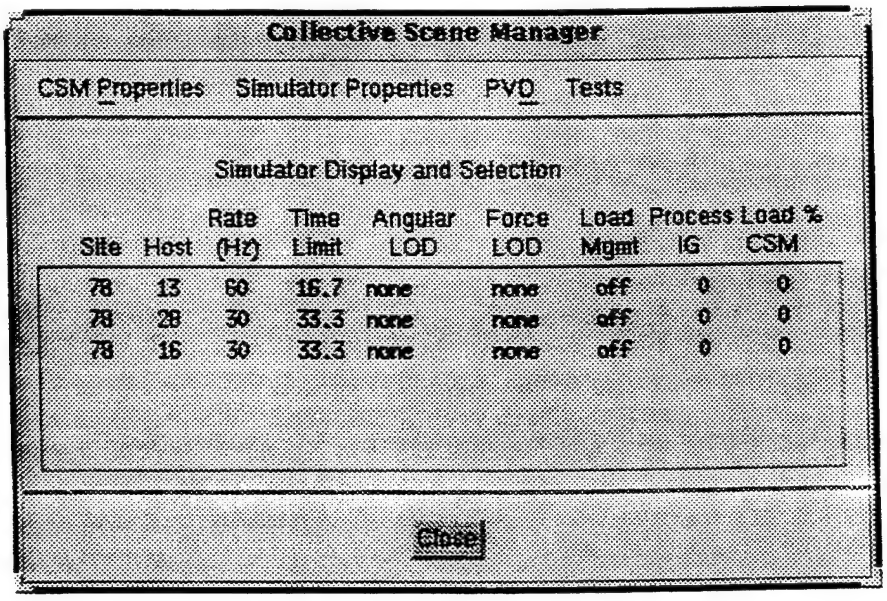


Figure 32 CSM Window at Startup

The CSM window displays fixed and variable information for each simulator being monitored. The window also contains several menus which the operator can select to display more information or to provide options for modifying CSM controls and behavior.

Site and Host numbers make up the first two parameters displayed for each simulator. In DIS terminology the Host number is actually the application number. The simulators IG update rate in Hertz or cycles per second appears next. The default time limit. in milliseconds. is the reciprocal of the rate. Angular and Force LOD controls are initially turned off, as is CSM Load Management processing. IG Process loads and CSM's prediction of the same are zero until a simulator is actually actively involved in an exercise.

8.1.2.2 CSM Properties

The first menu on the CSM window provides options for modifications that affect the entire CSM operation (as opposed to affecting only a specific simulator). The menu appears as is shown in Figure 33. Also shown are two windows that pop up if CSM Property options are selected.

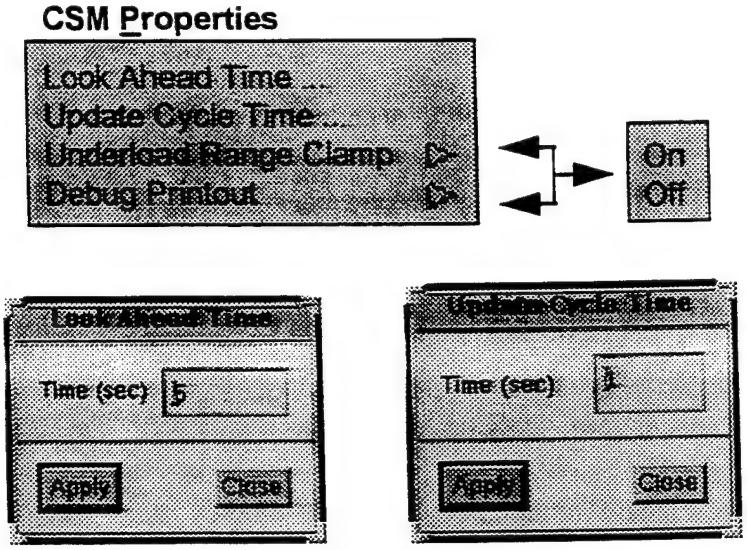


Figure 33 CSM Properties

Look Ahead Time refers to the time that CSM projects ahead and predicts where the ownship viewpoints and external models will be and from this what the processing loads could be. This allows CSM to be predictive rather than reactive so that it can prevent overload problems before they occur. The default look ahead time has been set to 5 seconds. Perhaps if the scenario involved fast moving air vehicles, the operator would choose to decrease the look ahead time. This is done by selecting the Look Ahead Time ... menu option, entering a new time (seconds) into the text box and selecting "Apply". The window can then be "Closed" to simplify the Exercise Manager display.

Similarly, the Update Cycle Time can be modified by the operator. This time is the interval CSM waits between its own processing time frames. Thus, if CSM can perform all of its predictive and corrective calculations and communications in less than the update cycle time (default is one second), it will pause until the time is up instead of immediately proceeding with more processing.

The other two CSM property options: underload range clamp and debug printout can be turned on and off by the operator. Debug printout is handy during development of new features; otherwise, it should be turned off to minimize screen text output which also can affect processing performance. The underload range clamp is used to suppress underload processing from occurring. This is also a debug tool that has not been used during the phase 2 project, but is maintained for possible use in the future.

8.1.2.3 Simulator Properties

The CSM operator can set and modify properties specified for selected simulators. A simulator is selected by clicking on its simulator line on the CSM main menu scroll list (See Figure 32). The Simulator Property menu items and sub-menu items are shown in Figure 34.

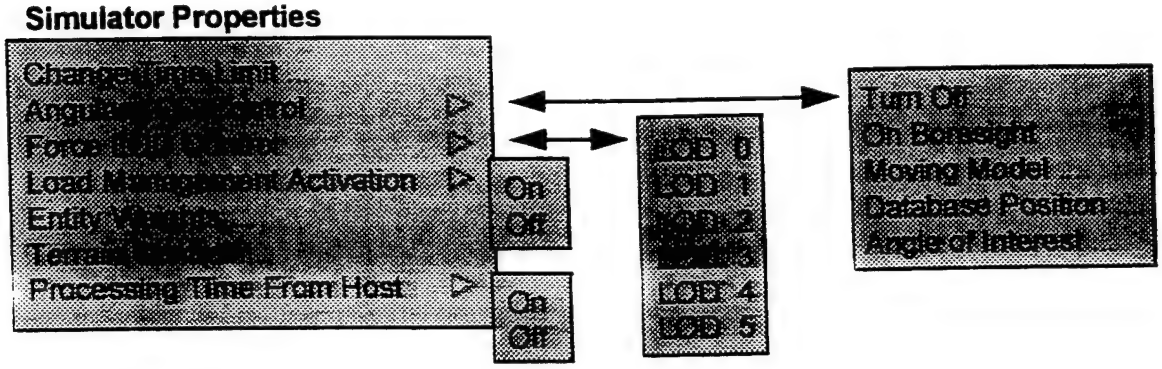


Figure 34 Simulator Properties Menu

The first item, Change Time Limit ..., when selected, causes a small window with a text box to appear as shown in Figure 35.

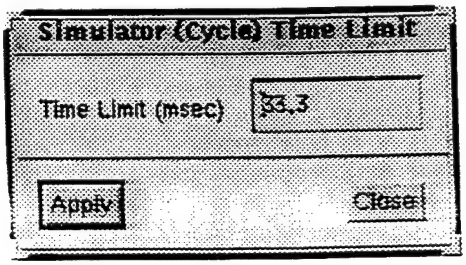


Figure 35 Simulator Cycle Time Limit

As mentioned previously, the initial time limit is related to the reciprocal of the IG cycle rate. CSM assumes there is no overload problem until this time limit is approached. The ability to decrease this time limit to force overload prevention measures to occur sooner is useful for demonstrating CSM capabilities when overload situations can not be achieved naturally. This adjustment can also be useful to tune CSM operations in an environment where CSM predictions are inaccurate. Examples for this include: changing the ownship field of view, or adding persistent special effects to the display scene.

8.1.2.3.1 Angular LOD Controls

Several controls are provided for making Model Level-of-Detail a function of angular displacement from some reference vector. The first sub-menu item, "Turn Off", will terminate any angular LOD control previously set for the selected simulator. The status, "off", would then appear in the main CSM window under Angular LOD for the simulator selected.

If "On Boresight" is selected. the reference vector is chosen to be that extending from the ownship position to the center of the observer field-of-view or center of the display channel. Moving models near the center of the display are shown at highest fidelity while models farther from the center are shown in simpler fidelities. Models near the display edge might not appear at all depending on what angle of interest has been set.

The final three menu selections each cause a popup window to appear to prompt the operator to provide more information. These three windows are shown in Figure 36.

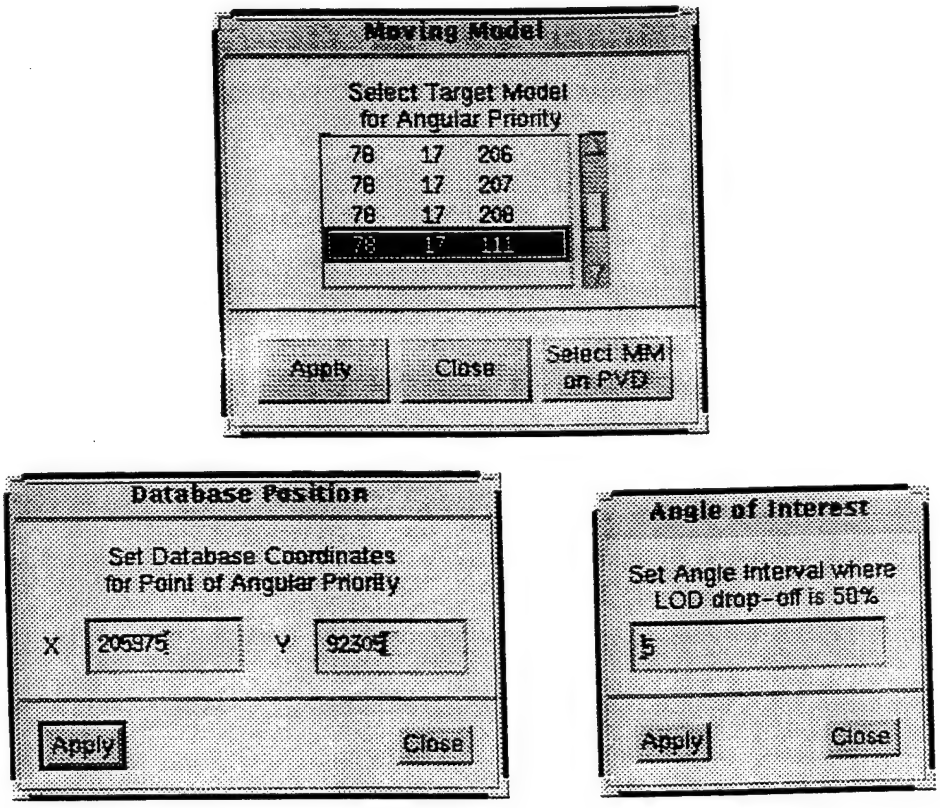


Figure 36 Angular LOD Control Windows

The Moving Model window presents a scrolled list of all external moving models being monitored by the selected simulator. After the operator selects a moving model and presses the "Apply" button, a reference vector between the ownship and the moving model will be computed. This reference vector will be used to determine displayed level of detail for each model in the scene. The selected model will, of course, be shown at highest fidelity. The option button, "Select MM on PVD" is used when the operator has selected a moving model from the Exercise Manager Plan View Display instead of selecting from the scrolled list. In either case the model ID will be highlighted in the scrolled list. In Figure 36, model 78:17:111 (site, application, entity ID) has been selected.

The Database Position window allows the operator to specify a specific point in the data base to serve as one reference vector vertex, the other being the ownship viewpoint position. The coordinates (X, Y), used by CSM are local to the simulator. The operator would typically choose a point strategic to the mission exercise such as the location of a bridge separating friendly and enemy forces.

The Angle of Interest window is used for the operator to choose the intensity of angular LOD control. The angle, in degrees, is entered into the text box and the "Apply" button is selected. The range scale factor used in model LOD selection becomes a cosine power function. Along the reference vector the range scale factor is unity. This factor is a half when the angle between the reference vector and the vector from ownship to moving model is equal to the angle of interest. If the angle of interest is large, model LODs are not as affected. If the angle of interest is small, most model LODs will be set such that the model is not seen at all unless the model is very near to the reference vector.

8.1.2.3.2 Force LOD Controls

The Force LOD control option allows the operator to set all moving models in the scene to a particular LOD value. The sub-menu allows for LOD 0 through LOD 5.

8.1.2.3.3 Moving Model Priority Weights

The Entity Weights option causes the pop-up window labeled "Moving Model Priority Weights" to appear in the Exercise Manager screen. See Figure 37. The scrolled list in this window identifies individual entities in the exercise and their current model priority weights. The operator selects an entity from the list or chooses the entity last selected from the Plan View Display, enters a new priority weight into the text box, and selects "Apply" in order to make a modification. The operator might change priority weights when the roles of friendly and enemy vehicles are reversed. Enemy vehicles should typically be of highest priority.

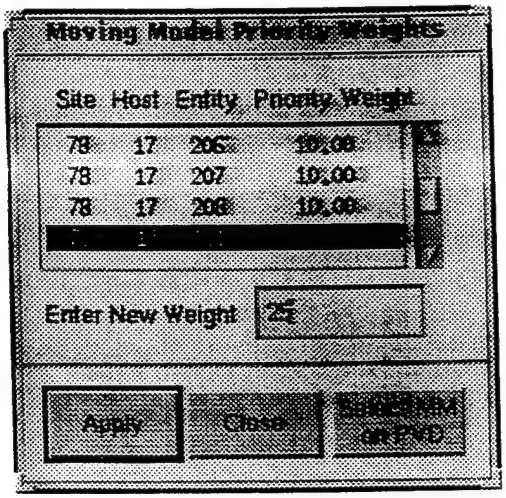


Figure 37 Moving Model Priority Weight Window

8.1.2.3.4 Load Management Activation and Terrain Control

CSM Load Management, for a selected simulator, is simply turned on or off using the sub-menu options. However, if terrain LOD control is to be done in conjunction with moving model LOD control, the terrain control pop-up window should first be activated, filled in, and applied. The operator indicates which type of database terrain, discrete or continuous, is loaded on the selected simulator, and enters a terrain priority value before applying. If the operator wishes to leave terrain LOD unaltered, the option, "Terrain Control OFF" should be selected.

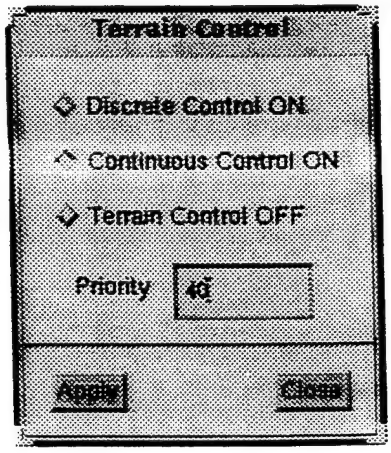


Figure 38 Terrain Control Window

8.1.2.3.5 Processing Time From Host

The operator can request the IG time used for processing from the host simulator via the on/off menu option. When activated, the host will report its processing load once every 5 seconds. This data is converted to a percentage by dividing by the Time Limit for the simulator.

Figure 39 shows a main CSM window display during typical operation. The IG process load % of the two active simulators (78:28 and 78:16) appear along with CSM predicted load percentages. The latter is determined from terrain load tables and pre-exercise measured loads of moving models at various LODs along with known ownship positions and current LODs of actual models in the scene. Load management is turned on for each active simulator. The "C" and "D" in "onC" and "onD" indicate that host 28 uses an continuous database and host 16 uses a discrete database. Inactive host 13 has angular LOD control set to "boresight" just to add some variety.

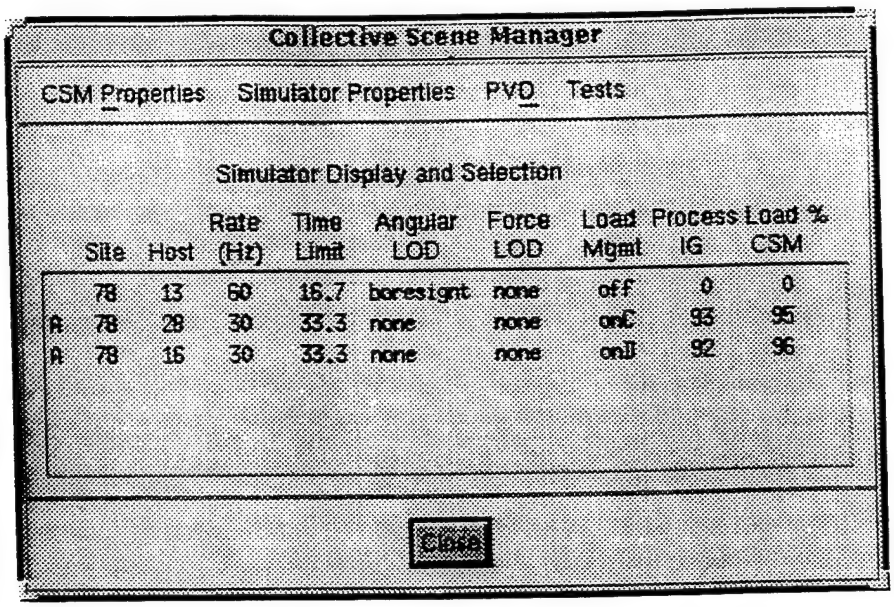


Figure 39 CSM Window with Active Simulators and Load Management

8.1.2.4 Plan View Display Tools

The Plan View Display tools discussed in section 4.2.3 are all activated or deactivated by on/off switches for the PVD menu items. Figure 40 shows the how the PVD menu items appear on the CSM window.

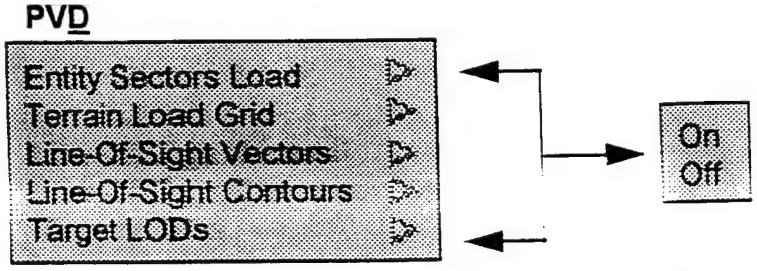


Figure 40 Plan View Display Tools Menu

8.1.2.5 The CSM Tests Menu

During CSM integration with the Host Simulator, it was useful to create simple tests to exercise all of the Set Data and Data Query PDUs that CSM was likely to send to the Host. Figure 41 shows the final appearance of the Tests menu which was changed and extended throughout integration testing. The last entry was actually a toggle of Error / Range values used to test control of continuous terrain.

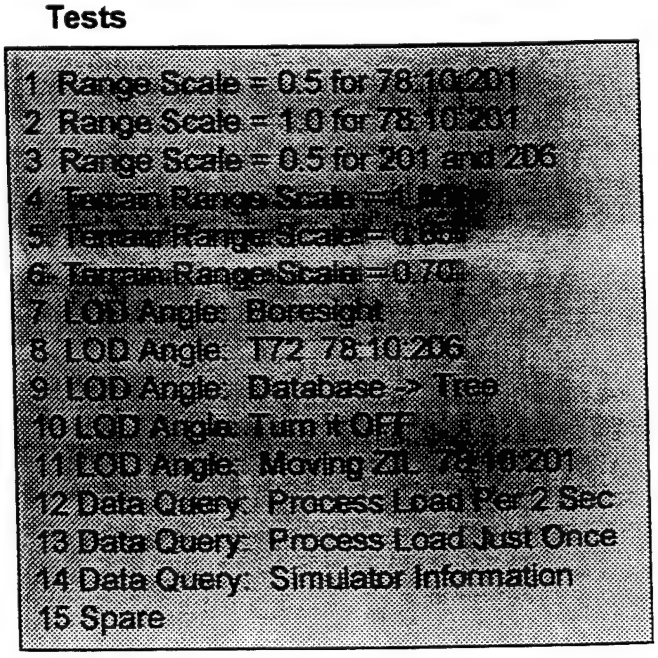


Figure 41 Tests Menu

8.2 Common Mission Functions

The software which comprises the common mission function package is discussed in the following sections. These sections describe the specifications and integration design which are necessary when attempting to interface to the software.

8.2.1 Code Specification and Software Integration

The mission function segment of code requires three specifications in its design. The first specification links the mission function layer to the calling routine which is typically the simulator or host. The second specification links the mission function layer to the database interface layer. The third and final specification links the database interface to the actual database. As illustrated in Figure 42, the user of this software, concerned with integration, must be primarily focused on the first and third specifications.

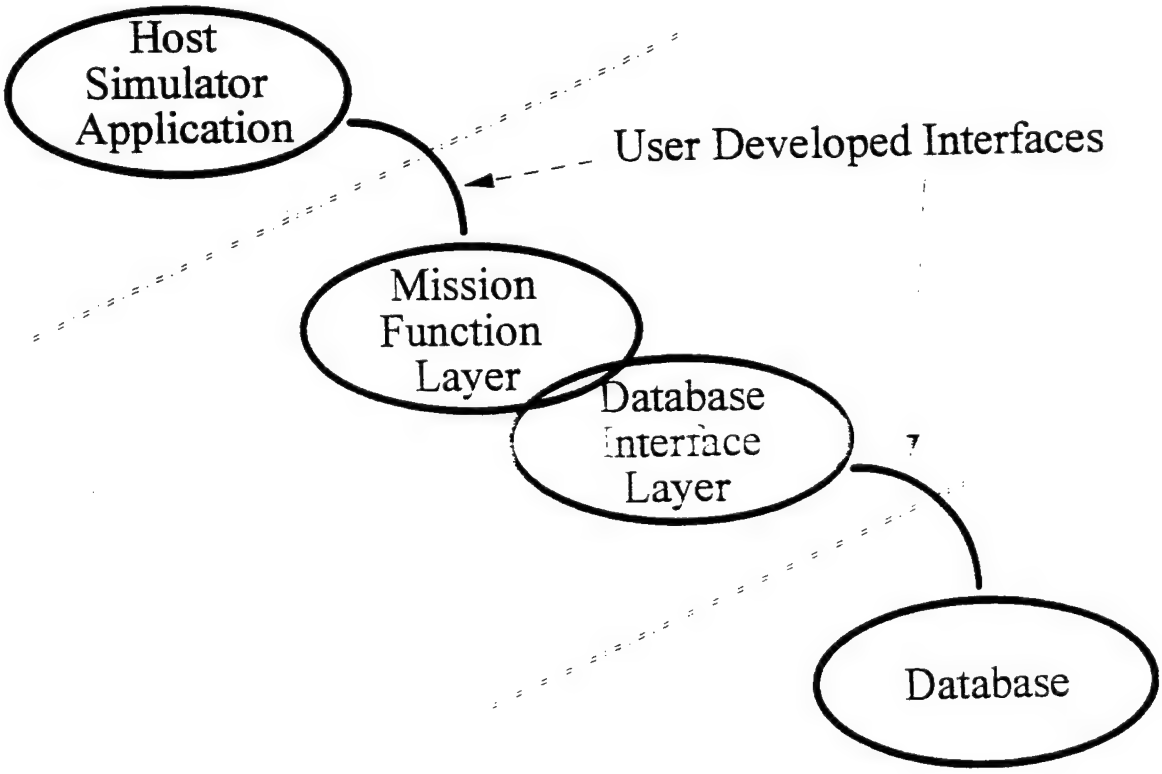


Figure 42. Common Mission Function Package Integration

The areas separated by dashed lines represent interfaces which must be developed using the specifications listed in the following section. Each of these areas will be discussed here.

8.2.1.1 Mission Function Layer To Host Specification

The ADA functions which comprise the mission function layer are listed here with a single line description of there function. From this specification as software developer can invoke any mission function which the package provides.

```
procedure Print_MF_Status(Status : in MFD.MF_Status); .
..... Print an error message based on the MF status

function MM_Load(MM_ID   : in MFMD.MM_ID_TYPE;
                 MM_Name : in MFD.MODEL_NAME) return MFD.MF_STATUS;
..... Add a moving model to model table

function MM_Unload(MM_ID : in MFMD.MM_ID_TYPE) return MFD.MF_STATUS;
..... Remove a moving model from model table

function MM_Update(MM_ID     : in MFMD.MM_ID_TYPE:
                   Num_Parts : in INTEGER_32;
                   Part_Data : in MFD.PART_DATA_ARRAY)
         return MFD.MF_STATUS;
..... Update moving model data in model table

 function MF_Define(MF_Req : in MFD.MF_REQ_TYPE) return MFD.MF_STATUS;
..... Define a new mission function


function MF_Update(MF_Req : in MFD.MF_REQ_TYPE) return MFD.MF_STATUS;
..... Update a mission function

 function MF_Terminate(MF_Req : in MFD.MF_REQ_TYPE) return MFD.MF_STATUS;
..... Terminate a mission function

procedure MF_Compute(MM_ID   : in  MFMD.MM_ID_TYPE;
                     MF_Type : in  MFD.MF_TYPES;
                     MF_ID   : in  INTEGER_32;
                     Results : out MFD.MF_RES_TYPE);
..... Compute a mission function

 procedure MF_Compute_Subcycle(Max_Time : in     FLOAT_64;
                               LM_Stats : in out MFD.MF_LM_STATS);
..... Processes current MF definitions

 procedure MF_IG_Result(IG_Results : in MFD.MF_RES_TYPE);
..... Puts IG results in results buffer

 procedure Return_MF_Result(Results : out MFD.MF_RES_TYPE;
                            Success : out BOOLEAN);
```

..... Returns a result from results buffer

```
function LM_Data_Update(Filter_Data : in EF.FILTER_UPDATE_TYPE)
      return MFD.MF_STATUS;
```
..... Updates filtering data for load management

```
procedure MF_Mode(Mode : in INTEGER_32);
```
..... Sets the mode in which the MF module operates

8.2.1.2 Database Interface Layer To Database Specification

The database interface specification is comprised primarily by the structures which hold the information. These structures are listed here for the reader's reference along with several procedures which comprise the database interface layer. All structures are defined with the exception of the MLBModel. This structure houses the model representation and is specific to the database which is being read. This structure becomes necessary if the user wishes to articulate parts. The current mission function package does not articulate parts.

Some elements are filled by the code. For example, the TranslatedFaces element of the MFModel structure is computed each time an update is made to a model position. Many of the elements in the MFDatabase structure are not used but are included for completeness. The only elements used within these structures are the database origin, the region sizes and the linked lists at the bottom.

Structures:

```
typedef struct
{ float  x;
  float v;
  float z;
} VectFloat;

typedef struct
{ float  distance_to_origin;
  VectFloat *edgePlaneNormals;
  VectFloat *vertices;
  int    face_is_terrain;
  float  number_of_vertices;
  VectFloat normal;
} MFFace;

typedef struct
{ MlbModel *mmod;
  int mmid;
  VectFloat LocNadPos;
  int Updated;
```

```
  MFFace *TranslatedFaces;
} MFModel;

typedef struct
{ MFFace *faces;  /* first face is terrain face, culture faces follow the
                     terrain face they sit on */
  VectFloat center;
  float    radius;
  int num_faces;
} MFCluster;

typedef struct
{ Vect coarseRegionCentroid;
  MFFace *faces;
  int    num_clusters;
  MFCluster *clusters;
} MFRegion;

typedef struct
{ int coordSysUnits; /* 1 ft, 2 meters */
  int coordSysType;
  float dbOriginLat;  /* degrees */
  float dbOriginLong;
  short latCoarseGridSize; /* in minutes of arc * 256 */
  short longCoarseGridSize;
  short latFineGridSize;
  short longFineGridSize;
  VectFloat minCorner;
  VectFloat maxCorner;
  float    regionRadius;

  char mlbFileName[256];
  int nMlbModels;
  MlbModel *mlbModels;  /* allocated */

  int firstGlobal;

  /* these are for the on-line component of the database */
  int xSizeOnline, ySizeOnline; /* long/lat in seconds */
  VectFloat cg;  /* center of gravity */
  float radius;  /* bounding radius */
  List *onlineRegionList;
  List *movingModelList;
} MFDatabase;
```

The only routine which must be supplied by the user for the database interface layer is a read database routine.

Routines:

read_database(*mission_function_database, dbOriginLat. dbOriginLong); ..... This routine interrogates the users database and returns the mission function structure listed above.

The rest of the routines inteface the DBI with the mission function module and are not listed here.

8.2.2 Users Manual

Once the package is integrated, the software suite is automated with necessary invocations from the host or simulator. This section then will describe the expected use of the functions within the common mission function package.

From the host, simulator or application calls are placed to the ADA package as directed in the specifications sections. Models can be loaded, moved and removed from the exercise using any of the MM_*process* procedures. Mission functions can be defined using the MF_Define procedure which allows the mission function to run every field from that point on. Defined mission functions can be changed (MF_Update) or removed (MF_Terminated) any time during the exercise.. If a mission needs to be computed only once then it is invoked by the MF_Compute procedure. If mission functions are required which are not provided with this package, the mission functions can be obtained from another process using the MF_IG_Result which allows a buffered mission function result transfer to take place.

From the database interface layer, a call is made to the user supplied readDatabase process upon a DBInit from the Mission Function Layer. Once the database is read in, the database interface section operates based on instructions from the mission function layer. It performs both 2D and 3D intersections for the mission function module. The 2D intersections is actual a misnomer, since it returns results from terrain intersections and culture intersections. The 3D intersections are intersections which occur with moving models. In this manner the mission function layer can determine which result is the desired result and operations such as terrain following bridges can be accomplished.

8.3 Terrain Load Software

The terrain load analysis software is a standalone executable application, called turfload, that communicates directly with the SE1000 image generator. It is run off-line and it produces a data file in ASCII format that is read in by the Collective Scene Manager at initialization time. The data consists of processing times for a grid of points in the target database. CSM uses this data, the viewpoint position, and bilinear interpolation to estimate the IG processing load for rendering terrain and culture.

8.3.1 Code Specification and Software Integration

The turfload application software is contained in one source file, surfload.cc, which is maintained in the baseline directory, /proj/dis/baseline/objlib/turfload/src. This code references many functions in libraries developed for the Exercise Manager and LMISC DIS software in general. In particular it uses communications functions to send/receive data to/from the Image Generator (IG). It tells the IG where to position and how to orient the viewpoint. It requests that the IG send turfload the current IG processing load. Turfload then waits to receive this data and write it to a file, load.dat. Below is a list of functions found in the surfload.cc source code. A short description of each function is given.

File surfload.cc

FUNCTION	DESCRIPTION
main	Create file load.dat. Call getProperties. Open net interface to Image Generator. Set up to receive buffer from IG. Read in the terrain database. Call EstimateSurfaceLoading.
getProperties	Read properties file chosen by operator. Extract and store: Latitude/Longitude boundary, number of grid rows and columns, LOD scale factors and range ring min and max ranges.
EstimateSurfaceLoading	Loop through grid rows and cols and scales. Compute viewpoint positions. Set IG LOD max and min ranges and Error/Range. Call EstimateRegionLoading.
EstimateRegionLoading	Use terrain following functions to position and orient ownship view. Raise viewpoint 20 feet. Set viewpoint position in IG. Delay. Set heading in IG for each angular sector. Request process load statistics from IG. Wait. Store Frame 2 processing time in milliseconds to load.dat.
GetLine	Extracts line from text file. Ignores comments.
ReceiveIGBuffer	Called per IG frame. Dump buffer contents.
WaitForSync	Wait for specified number of frames.
createProjectile	Just a stub.
lwrCase	Not Used Here.

Turfload reads in a text properties file which contains the following:

Terrain type (0 = discrete, 1 = continuous)
Grid boundaries (min and max latitudes and longitudes)
Number of rows and columns that make up the grid.
Sector angular width (data taken at various headings for each grid point)
Number of LOD adjusting Scale levels and actual scale factors
Number of LOD Range Rings (min and max ranges for an LOD)
and the actual min and max ranges for each range ring (discrete only).

8.3.2 Users Manual

To run turfload:

(1) start up the SE1000 image generator; get an OIC GUI. Load a database.
(2) log in with root privileges.
(3) set directory for executable (cd /proj/dis/baseline/build/turfload/sun_os5).
(4) invoke the executable (./turfload)
(5) select a properties file from list displayed.
(6) select OIC button, "enable host"

Depending on the grid resolution specified in the properties file, turfload could take several hours to complete its data gathering. When finished, the user should rename the output load.dat file and move it to a directory where it will be read in by CSM. The path name should be placed in sim_niu.list.

9. CONCLUSIONS

The inclusion of terrain LOD adjustment proved to be extremely valuable to the CSM algorithm. It provided a mechanism for the CSM to compensate for overload without adjusting the look of the players or entities. This method of compensation proved to work extremely well in the feasibility study and demonstration which was given during the last month of the project. Terrain LOD was adjusted in this demonstration prior to the vehicles entering the field of view. Furthermore, in the continuous terrain case, only one moving model required to be rendered at a lower level of detail beyond the terrain adjustments to overcome IG overload.

During the software development phase, the terrain load analysis effort raised a number of questions during its design phase. Since the CSM operates in a predictive fashion, a-prior knowledge of loads of both terrain and models is important. This information is currently obtained by off-line (non-real-time) processes and used by the CSM during the exercise. Many issues arose during the collection of this information. Extending the terrain load information to apply to air vehicle is a good example. Air vehicles, like helicopters, introduce a fifth degree of freedom into the data construct which comprises the terrain load information. This fifth degree of freedom is pitch or downlook angle. This greatly increases the size of the terrain load table. If the air vehicle happens to be a fixed wing aircraft, roll will probably introduce yet another field into this data. If all possible factors which affect the terrain load data were brute forced into a table, the table would require seven indices to access the data. This seems to indicate that alternative methods of terrain load analysis should be investigated.

The integration of the CSM to a PVD provided mechanisms with which a set of tools could be developed to help detect and measure interoperability problems. These tools are a preliminary process for automatic detection of interoperability issues. If detection of interoperability problems could be automated and documented somewhere, for instance, in an after action review, then these ogs of real interoperability issues could be investigated to help provide solutions to them. In the least, a mechanism is in place which, during distributed simulation, would point out the problems associated with dissimilar simulator environments. These points could then be considered when performing the AAR.

The common mission function package provides a set of software which could be distributed and integrated into simulators and hosts to yield correlated mission function computations throughout a distributed exercise. Integration difficulty depends heavily on the application and the database but is never unattainable.

The feasibility study for the CSM conveyed positive results. The CSM behaved as expected and fully compensated for a overload situation. An interesting extension to this study would be a large scale DIS application. By large scale, it is meant several simulators of varying types such as SGI, E&S and Lockheed Martin as well as ModSAF and Exercise Manager CGF forces. This would be a time consuming study but would assess feasibility of the CSM design in a much more rigorous manner.

10. REFERENCES

10.1 Government Documents

1. Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications, Version 2.0 Third Draft, Institute for Simulation and Training, 12424 Research Parkway, Suite 300, Orlando, FL 32826.

2. Map Projections - A Working Manual by John P. Snyder, US Geological Survey Professional Paper 1395, United States Government Printing Office, Washington, DC, 1987.

10.2 Non-Government Documents

1. DIS Demonstration Testbed for I/ITSEC 1993, by R. L. Ferguson and E. Pollak, Martin Martin Marietta Information Systems Company, November, 1993.

2. Research and Development of Terrain Data Bases for Distributed Interactive Simulation, Volume I, Technical Proposal, submitted to STRICOM in response to STRICOM & ARI Broad Agency Announcement BAA NTSC 93-02 by the Institute for Simulation and Training.

3. IRIX Performer Programming Guide, Silicon Graphics, Inc., 1992.

4. Compu-Scene PT2000 Real-Time Operations Manual, Martin Marietta Information Systems.

5. Distributed Interactive Simulation, Testing Handbook, July 27, 1993, Version 1.0, Institute for Simulation and Training.

6. ITSEC DIS Interoperability Demonstration Test Procedures and Results, for contract N61339-91-C-0103 for STRICOM, February 16, 1993, Institute for Simulation and Training.

7. Geographic Position Accuracy in a Distributed Simulation Environment, by Dr. Richard Economy, Robert Ferguson, Dr. Eytan Pollak, Martin Marietta Information Systems, to be published in Image VII, June 1994.

8. Computer Generated Forces Testbed, Release 6.400, Institute for Simulation and Training.

9. DIS/CSM-Collective Scene Management, a Video Tape from Martin Marietta, 25 June 1994.

10. "Enhancement and Use of the IST CGF Testbed for a Precision Gunnery Demonstration," 4th Conference on Computer Generated Forces and Behavioral Representation, May 1994.

11. "Interoperability Of Dissimilar Visual Systems. A Prototype Scene Manager" Final Results Report. Lockheed Martin. 30 June 1994.

12. "A Prototype DIS Scene Manager". Michael Tackaberry, Dr. Eytan Pollak, Anthony Pelham, Martin Marietta Information Systems. 11th DIS Workshop September 1994.

13. "Standard for Distributed Interactive Simulation -- Application Protocols", IEEE 1278.1-1994.

14. "Enumeration and Bit-encoded Values for use with IEEE 1278.1-1994, DIS Application Protocols" 1994.

15. "Interoperability Of Dissimilar Visual Systems: An Enhanced DIS Scene Manager" Interim Technical Report, Dr. Eytan Pollak, William Mayes, Michael Tackaberry, William Garbacz, Dr. Thomas Wilson, Lockheed Martin Information Systems, January 15, 1996.

16. "Distributed Interactive Simulation Exercise Manager". Dr. Eytan Pollak, Sandra Vaquerizo, Lockheed Martin Information Systems, 1996 Image Conference, June 1996.

17. "Interoperability Of Dissimilar Visual Systems By Scene Management", Dr. Eytan Pollak, William Mayes, Mike Tackaberry, Dr. Tom Wilson, Lockheed Martin Information Systems. 1996 Image Conference, June 1996.

APPENDIX A: ABBREVIATIONS AND ACRONYMS

AAR	After Action Review
API	Application Programmer's Interface
CIU	Cell Interface Unit
CGF	Computer Generated Forces
CMF	Common Mission Functions
CSM	Collective Scene Manager
DEC	Digital Equipment Corporation
DBI	Database Interface
DIS	Distributed Interactive Simulation
FOV	Field Of View
GP	Geometry Processor
GUI	Graphical User Interface
Hz	Hertz or Cycles per Second
IEEE	Institute of Electrical and Electronics Engineers, Inc.
IG	Image Generator
I/ITSEC	Interservice/Industry Training Systems and Education Conference
IR&D	Internal Research and Development
IST	Institute for Simulation and Training
IVACC	Interoperable Visuals for Air Combat Command
LAN	Local Area Network
LMISC	Lockheed Martin Information Systems Company
LOD	Level of Detail
LOS	Line of Sight
ModSAF	Modular Semi-Automated Forces
MFM	Mission Function Module
ms	Milliseconds
NE	Number of (Pixel) Elements
NIU	Network Interface Unit
PC	Personal Computer
PDU	Protocol Data Unit
PVD	Plan View Display
SAFOR	Semi-Automated Forces
SGI	Silicon Graphics, Inc.™
SIF	Standard Interchange Format
SIMAN	Simulation Management (PDUs)
STRICOM	Simulation, Training & Instrumentation Command
UDP/IP	User Datagram Protocol / Internet Protocol
UI	User Interface
VP	View Point